1992-06

A combinatorial approach to automated LOFARGRAM analysis

Brahosky, Vance A.

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/26922

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE		Form Approved OMB No 0704-0188
1a REPORT SECURITY CLASSIFICATION: Unclassified	1b RESTRICTIVE MARKINGS	
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited.	
2b DECLASSIFICATION/DOWNGRADING SCHEDULE		
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)	
6a NAME OF PERFORMING ORGANIZATION: Naval Postgraduate School; 6b OFFICE SYMBOL (If applicable)	7a NAME OF MONITORING ORGANIZATION: Naval Postgraduate School	
6c ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000	7b ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000	
8a NAME OF FUNDING/SPONSORING ORGANIZATION; 8b OFFICE SYMBOL (If applicable) 33	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO; TASK NO; WORK UNIT ACCESSION NO	

11 TITLE (Include Security Classification)
A COMBINATORIAL APPROACH TO AUTOMATED LOFARGRAM ANALYSIS

12 PERSONAL AUTHOR(S)
Brahosky, Vance A.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM ______ TO ______	1992, June 18	90

16 SUPPLEMENTARY NOTATION The views in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	LOFARGRAM; Graph Theoretic Tracker(GTT); Hough Transform; Heuristic Search; Cluster Analysis; Feature Space; Parameter Space

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

This thesis examines the combination of three algorithms: Graph Theoretic Tracker (GTT), Hough Transform, and Heuristic Search to enhance the detection of spectral tracks of underwater targets in LOFARGRAMS. Previous studies examined these algorithms separately. Here, GTT is used as a pre-processor of the LOFARGRAM display data to locate optimum paths of signals through noise. The line tonals in the output image from GTT are then manipulated by the Hough Transform into clusters of points in parameter space. A Heuristic Search sorting technique is employed to determine cluster centers. These cluster centers are then reconstructed back into line tonals using the inverse Hough Transform formula. The results of this thesis show improvements by taking the Hough Transform of the original LOFARGRAM masked by the output of GTT. The effect of background noise is offset by the accumulation in the parameter space. Subsequently, the recovery of desired tonals is improved.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION	
[✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT ☐ DTIC USERS	Unclassified	
22a NAME OF RESPONSIBLE INDIVIDUAL: Chin-Hwa Lee	22b TELEPHONE (Include Area Code): (408) 646-2190	22c OFFICE SYMBOL: EC/Le

DD Form 1473, JUN 86 — Previous editions are obsolete — SECURITY CLASSIFICATION OF THIS PAGE

S/N 0102-LF-014-6603 — Unclassified

A Combinatorial Approach to Automated LOFARGRAM Analysis

by

Vance A. Brahosky
Lieutenant , United States Navy
B.S., Pennsylvania State University, 1985

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 1992

ABSTRACT

This thesis examines the combination of three algorithms: Graph Theoretic Tracker (GTT), Hough Transform, and Heuristic Search to enhance the detection of spectral tracks of underwater targets in LOFARGRAMS. Previous studies examined these algorithms separately. Here, GTT is used as a pre-processor of the LOFARGRAM display data to locate optimum paths of signals through noise. The line tonals in the output image from GTT are then manipulated by the Hough Transform into clusters of points in parameter space. A Heuristic Search sorting technique is employed to determine cluster centers. These cluster centers are then reconstructed back into line tonals using the inverse Hough Transform formula. The results of this thesis show improvements by taking the Hough Transform of the original LOFARGRAM masked by the output of GTT. The effect of background noise is offset by the accumulation in the parameter space. Subsequently, the recovery of desired tonals is improved.

C,

TABLE OF CONTENTS

I. INTRODUCTION

A. BACKGROUND

As sonar systems continue to increase in size and complexity, operators are in danger of being overwhelmed by the information presented in displayed data. There is a need for algorithms that can support automatic detection and tracking of signals of interest. In military applications, improvements in the area of detection of signals in noise are always of continuing interest. Concerning Anti-Submarine Warfare (ASW), one classical method of displaying passively received acoustical data is the LOFARGRAM. LOFAR is an acronym for Low Frequency Analysis and Recording. In the form of a waterfall display, acoustical data is presented spectrally with the y-axis representing time and the x-axis representing frequency. In the frequency domain, man-made signals possess a higher power content when they are compared to background noise. Because the shading of the pixels in the LOFARGRAM is power content dependent, pixels with higher power appear brighter.

B. PREVIOUS WORK

A broad spectrum of techniques have been applied to detect acoustic signals passively in ocean noise. Two previously studied algorithms include the Graph Theoretic Tracker (GTT) and the Hough transform. Both algorithms have been examined in detail using synthetic test sets. Ross [Ref. 1] gives a detailed analysis of GTT while Wang's study is devoted to the Hough transform [Ref. 2]. The main advantages of these algorithms include: requiring no *a priori* knowledge of location or numbers of signals to detect; having the ability to detect single, multiple, crossing, and swept tonals; producing output suitable for immediate display; and providing accurate frequency estimates. The study here is aimed at improving the overall performance of tonal tracking.

C. THE DESCRIPTION OF THE PROBLEM

This thesis investigates the application of the GTT and Hough transform algorithms on a real data set provided by the Naval Research Laboratory (NRL). The use of synthetic data sets in previous studies allow tighter control of experimentation and is therefore well suited for quantitative analysis. However, in this thesis, the performance of these algorithms using real data provides a more accurate qualitative evaluation.

A tandem combination of GTT and Hough is used to maximize their respective advantages and minimize their inherent disadvantages. A final step involves the use of a heuristic search sorting technique. This step determines which clusters in the parameter space of the Hough transform should be reconstructed back into line tonals in the feature space.

BEAM00.BIN is a LOFARGRAM provided by NRL. Displayed in Figure 1, BEAM00.BIN is a 256x256 byte image of three stable tonals and one swept tonal at a low signal-to-noise ratio (SNR). In order to keep this thesis at the "UNCLASSIFIED" level, both the origin and frequency content of this LOFARGRAM are not discussed and in fact are unknown to the author.

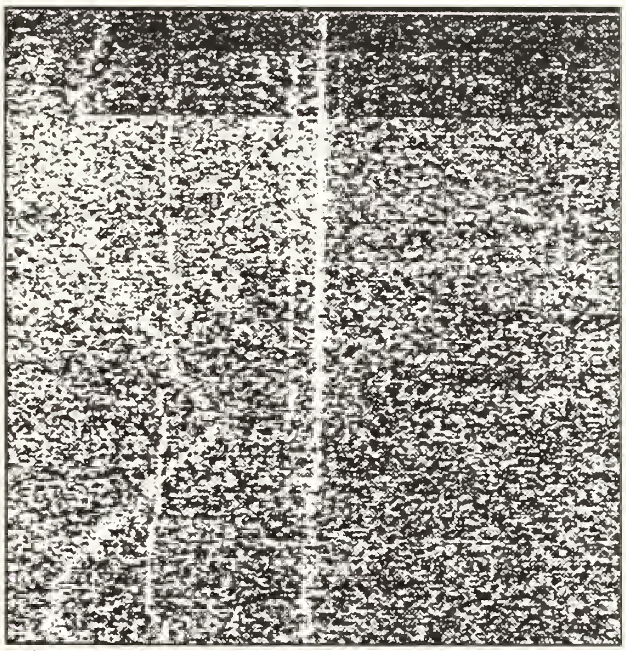
Figure 1. LOFARGRAM BEAM00.BIN

The direction of this thesis is best illustrated by the flow graph presented in Figure 2. Because the GTT algorithm operates most efficiently with normalized data sets, BEAM00.NRM, the normalized version of BEAM00.BIN, is used as the input LOFARGRAM into GTT.

The output file from GTT, TRACKS.B, is then utilized as input for the Hough transform. After the image has been transformed into clusters in parameter space, a heuristic search sort is performed to determine which clusters are suitable for reconstruction. Finally, the inverse of the Hough transform is employed to convert selected clusters back into line tonals.

This thesis consists of five chapters. Chapter I provides an introductory description of the track detection problem. Chapter II presents a description and implementation of the GTT algorithm. Chapter III emphasizes the Hough transform method. Chapter IV details both the original and improved heuristic search algorithm. Finally, Chapter V includes conclusions and recommendations for further study.

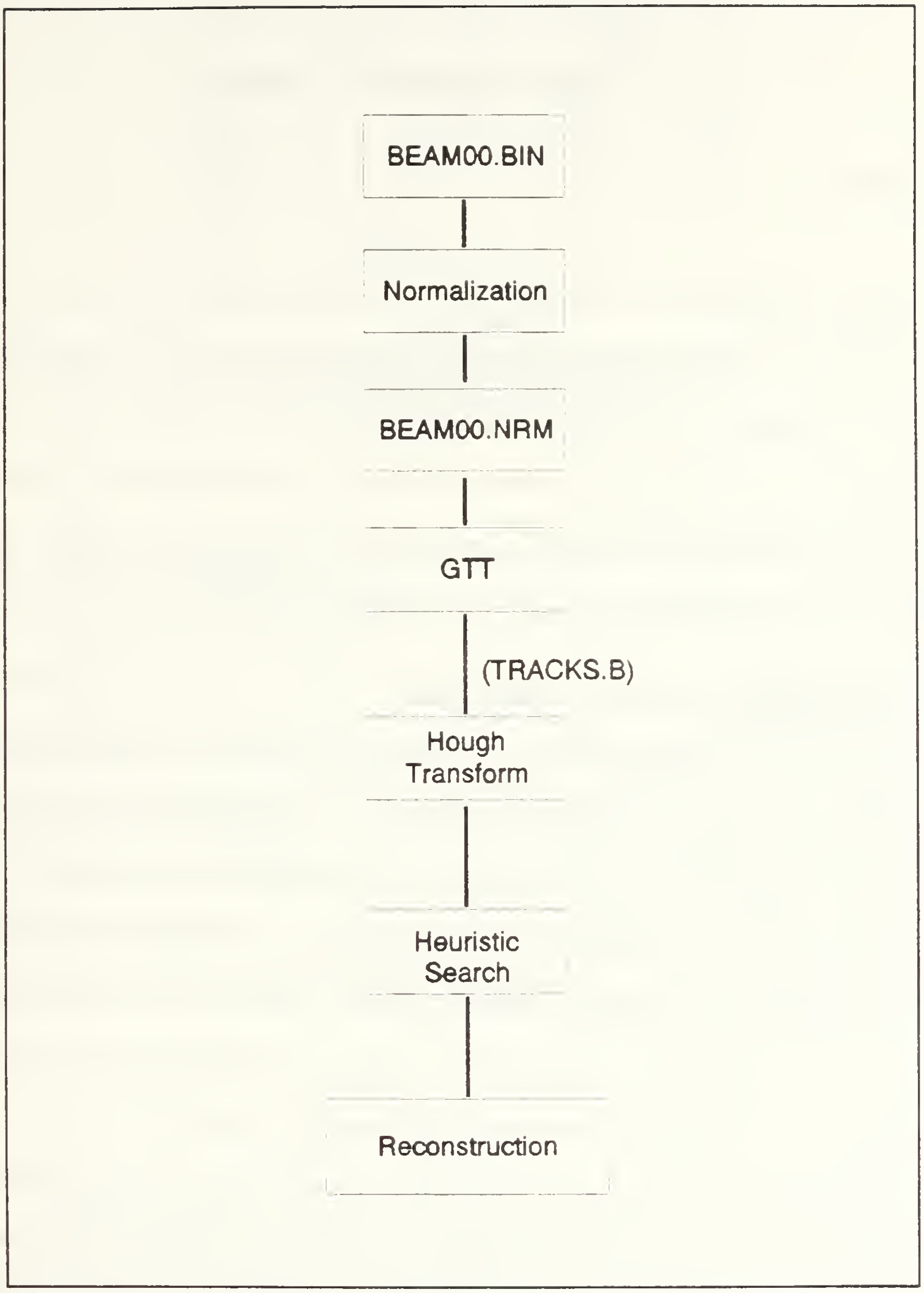

Figure 2. Procedure Flow Graph in this Thesis

II. GRAPH THEORETIC TRACKER

A. THEORY

Graph Theoretic Tracker (GTT), developed by L.J. Wu and R.A. McConnell at Naval Research Laboratory (NRL), Washington, D.C. , is based on graph partition theory [Ref. 3]. Jensen developed this optimum network partitioning theory whereby a set of elements are partitioned in such a way that a pre-defined cost function is optimized [Ref. 4].

B. DEVELOPMENT OF THE ALGORITHM

Applied to LOFARGRAM analysis, the track detection problem is simply translated into a graph partitioning problem. Optimum partitioning is utilized to extract features such as prominent line tonals. Each time line of the LOFARGRAM is mapped onto a graph where the individual pixels or frequency bins of the image correspond to nodes in the graph. According to Wu and Curtis [Ref. 3],

> ...the partitioning is accomplished through the orderly enumeration of all possible partitions of a graph followed by a recursive search using dynamic programming methods. The final partition generated by this algorithm is optimal with respect to some objective cost function used to drive the dynamic programming search.

Sample partitions of a graph are shown in Figure 3.

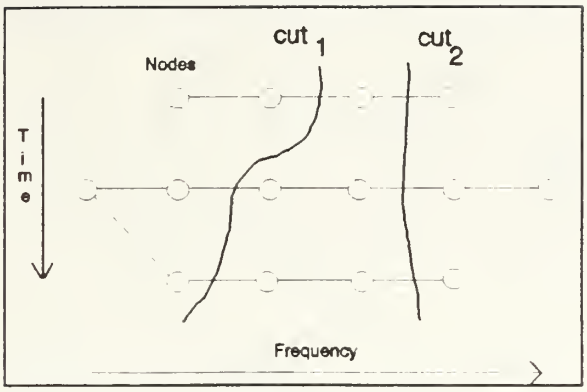

Figure 3. Translation of LOFARGRAM to GTT Graph

In order to maintain connectivity requirements, two dummy nodes are used as end nodes. Line tonals in the LOFARGRAM will appear as cuts through the graph. In order to detect these tonals, the optimum partitioning algorithm utilizes a cost function based on the signal-to-noise ratio (SNR). The noise estimate needed for this cost function is found by calculating the mean weight of the nodes between pairs of tracks or cuts. This is shown graphically in Figure 4.

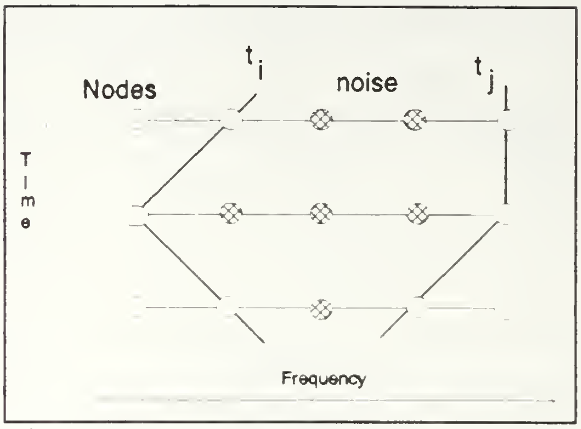

Figure 4. Cost Function in GTT

Symbolically, the cost function is defined as follows:

$$cf_{ij} = \gamma\sigma_{ij} - \eta_i, \qquad (2,1)$$

where the signal estimate, η_i, is determined by integrating the graph weights along the track path, $\mathbf{t}_i$, and the noise estimate, σ_{ij}, weighted by a scalar constant, γ, is obtained by calculating the mean weight of the nodes between $\mathbf{t}_i$ and $\mathbf{t}_j$; the scalar constant, γ, can be used as a means to set detection threshold. The cost function, $\mathbf{cf}_{ij}$, is simply the cost of a particular pair of cuts through the graph. By summing all possible cuts, the total cost can be determined:

$$totalcost = \sum_{i} cf_{ij}. \qquad (2.2)$$

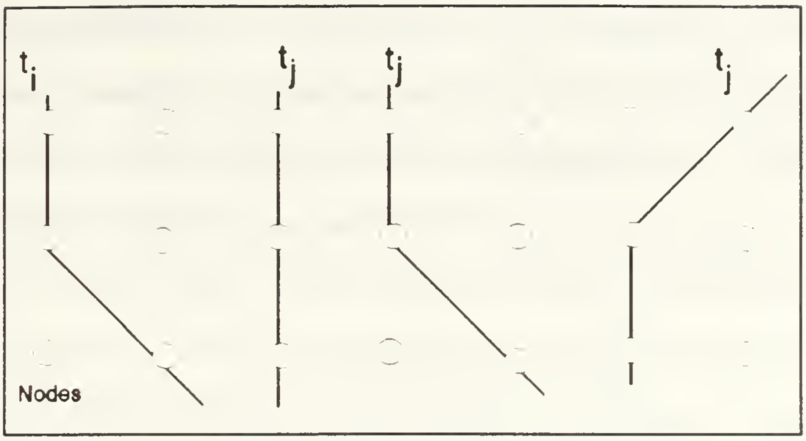

Figure 5. Various Possible Tracks

C. IMPLEMENTATION OF GTT

1. Parameters Involved

Because the number of potential signals in a typical LOFARGRAM can be quite large, graph partitioning processing times can become unreasonable. By limiting the number of time lines of data processed, GTT can produce output in an acceptable time period. This is achieved by limiting the height and width of the processing window. Currently the variables used to control this window include K, L, and P. K establishes the maximum number of pixels or frequency bins that a track can deviate from one time line of data to the next. A maximum value of four is allowed for K indicating that a track has nine possible positions in the succeeding line, {-4,-3,-2,-1,0,+1,+2,+3,+4}, relative to its current position.

The parameter L determines the height of the processing window, limiting the number of lines of data to be partitioned. A maximum of three lines is permitted. P defines the width of the processing window and represents the number of frequency bins or pixels per time line. The maximum value allowed for this parameter is 512. By using these constraints, the number of possible cuts, **N**, evaluated reduces to

$$N = P(2K + 1)^{L-1} << 2^{LP-2}. \qquad (2.5)$$

For comparison, Table 1 shows the number of cuts analyzed for a 2x256 window of data from BEAM00.BIN with the number of cuts performed if the maximum values for all parameters is used.

Table 1. COMPARISON OF CUTS

2x256 window of data	maximum values allowed
P = 256	**P** = 512
K = 1	**K** = 4
L = 2	**L** = 3
N = 768	**N** = 41472

2. Source Code

The software implementation of GTT, provided as Appendix A, contains the following programs, listed in the order in which they are used.

- MAIN.C
- PART.H
- INITIAL.C
- GENNODE.C
- UPDATE.C
- PARTIT.C
- LIBRARY.C

In order to take full advantage of modularity inherent to the C programming language, the GTT algorithm is implemented as a main program, MAIN.C, calling five separate source files. PART.H is provided as a header file to establish definitions of and limitations on the variables used. A graphical depiction of GTT activities is shown in the flow chart provided in Figure 6.

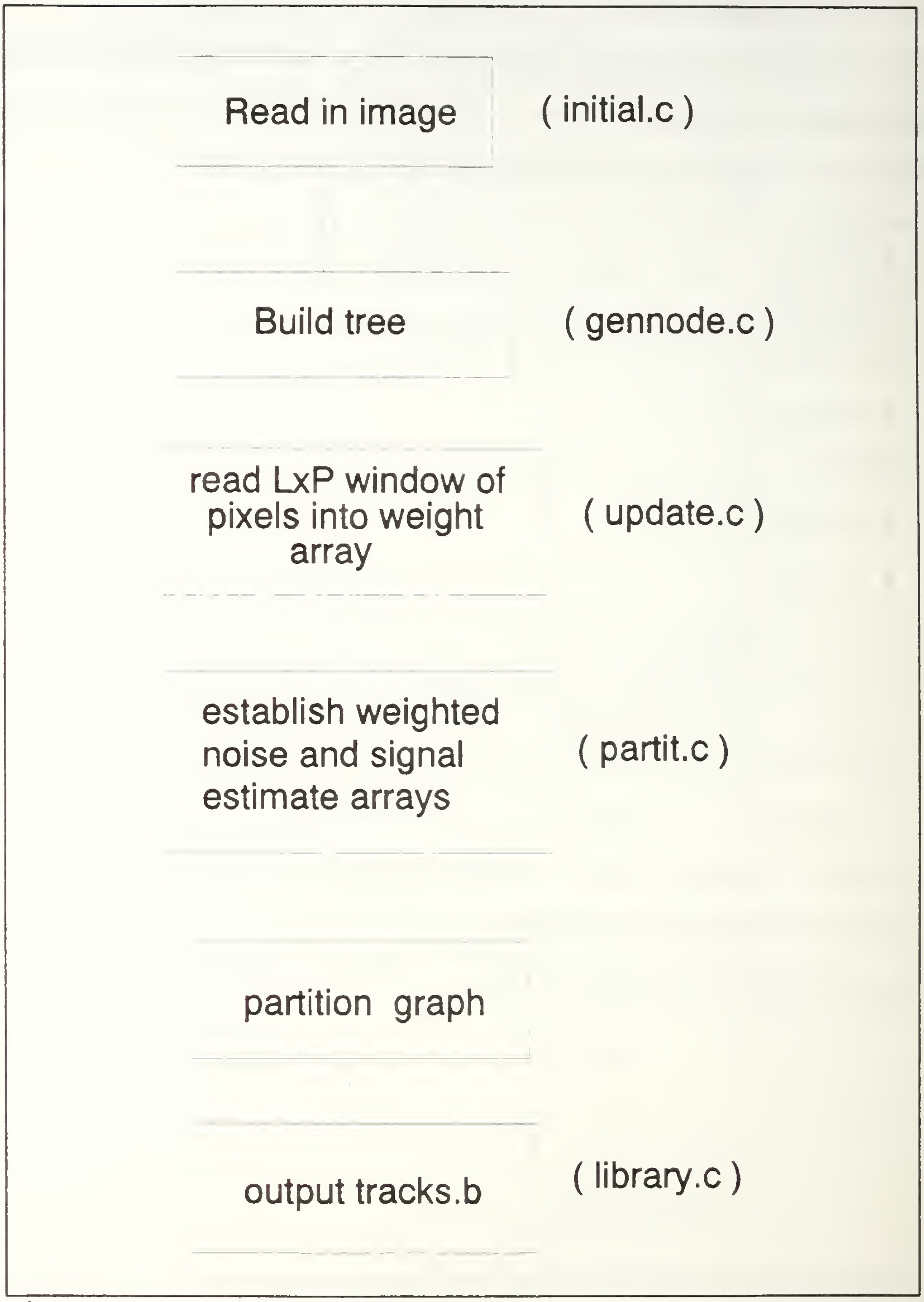

Figure 6. GTT Flow Chart

Using command line inputs, INITIAL.C determines if the parameter K, L, and P are within required limits and reads the LOFARGRAM in image format. GENNODE.C is then called to build the tree that will be graph partitioned. This module maps individual frequency bins in the display image onto nodes in a graph. In the main loop of MAIN.C, GTT successively evaluates an LxP window of data as a graph to be partitioned. Within this window, GTT attempts to maximize the total signal while minimizing the noise. The output from this window is a set of cuts that represent the best potential tracks. Within the main loop, UPDATE.C is called to read this LxP window of pixels into a weight array. Graph weights for the nodes are determined by the pixel intensity representing those nodes. UPDATE.C then constructs arrays for the weighted noise estimate and signal estimate. When this is completed, PARTIT.C is called to partition the graph based on the cost function (see equation 2.2). Finally, LIBRARY.C receives the LxP partitioned graph and outputs this as TRACKS.B. The main loop of MAIN.C is traversed again using the next LxP window of data from the input file. The output file, TRACKS.B, is displayed in the same format as the input LOFARGRAM, which is a waterfall display of frequency versus time. Contrary to the input file which is displayed in 256 gray levels, TRACKS.B is binary with the black background representing the absence of signal and white portraying the presence of signal.

For clarity in display, the inverse of the image of TRACKS.B is used in the following figures. The GTT output of BEAM00.BIN, shown in Figure 7, was obtained with the parameter settings of **K=1**, **L=2**, and **P=256**. Although the GTT output of BEAM00.BIN displays a large number of false alarms or noise, the three constant tonals and the swept tonal are visibly evident.

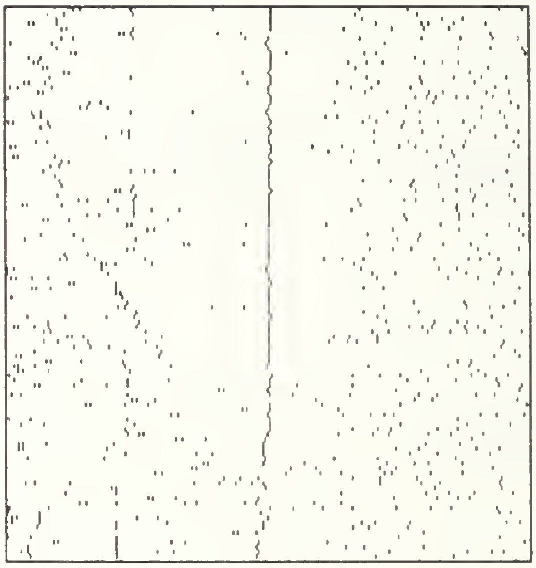

Figure 7. GTT Output, TRACKS.B, with $\gamma=6$

3. BEAM00.NRM Image

The LOFARGRAM BEAM00.NRM is the normalized version of BEAM00.BIN. BEAM00.NRM is used as the input data set for GTT. To determine the pattern of normalization, the IM-4000 Image Manager is used to analyze the histograms of both BEAM00.BIN and BEAM00.NRM. Containing the full spectrum of gray scales 0-255, BEAM00.BIN possesses a mean pixel value of 161.

BEAM00.NRM however is normalized to 64 gray scales with a mean pixel intensity of seven. In order to reproduce BEAM00.NRM, the following standard normalization equation is used:

$$Norm\_Imag = Old\_Imag * \left(\frac{255}{\max\_pixel - \min\_pixel}\right). \qquad (2.6)$$

GTT however does not function properly with the normalized image. Further study reveals that BEAM00.NRM is probably constructed by normalizing each record or time line of data instead of global normalization of the entire image. Instead of analyzing this normalization further, this thesis concentrates on manipulating the output of GTT, using BEAM00.NRM as input.

4. Detection Threshold

Mentioned earlier, the detection threshold can be manipulated by changing γ in the cost function calculation, equation (2.1). This can be achieved by numerically changing the scalar constant used in the function, **COSTFUNCTION(i,j)**, of **PARTIT.C**. Figures 8, 9, 10 , and 11 show the effects of varying γ from five to ten.

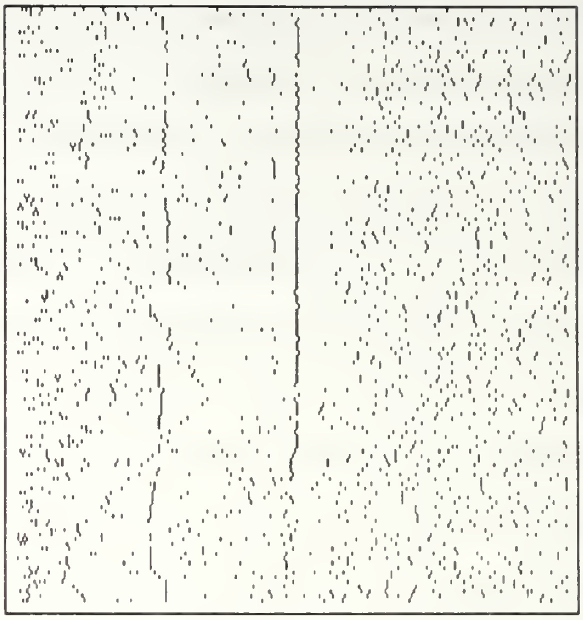
Figure 8. Effect of Threshold
$\gamma=5$

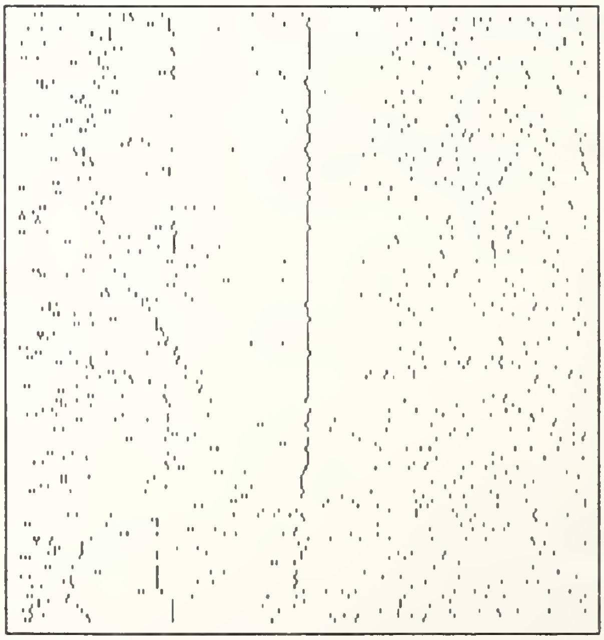
Figure 9. Effect of Threshold
$\gamma=6$

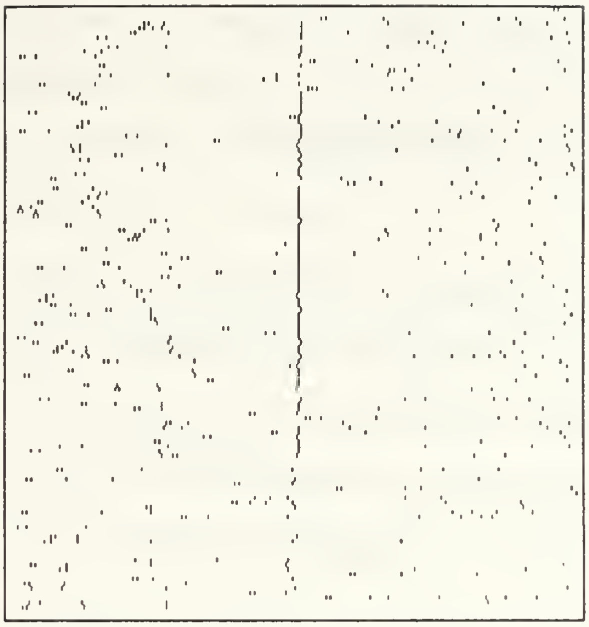
Figure 10. Effect of Threshold
$\gamma=7$

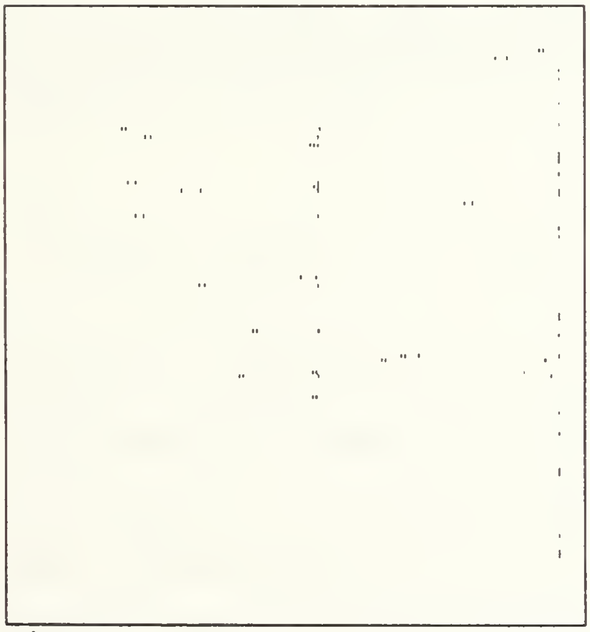
Figure 11. Effect of Threshold
$\gamma=10$

As seen in these figures, the decision to set a detection threshold, γ , at a certain level involves trading off detection for false alarms. If the threshold is set at a low value, as it is in Figure 8, the signal becomes more evident at the expense of introducing a large number of false alarms. As the threshold is increased to values of six, seven and finally ten in Figures 9, 10, and 11, the number of false alarms diminishes but the probability of miss increases. From visual observation, a detection threshold of six provides the best output.

The output of GTT shows the detection of prominent points along the track, but those points tend to be discontinuous. The next processing step involving the Hough transform is intended to detect potential line tonals existing in the output of GTT. The Hough transform achieves this by determining which disconnected points exhibit collinear tendencies.

III. HOUGH TRANSFORM

A. THEORY

One method of detecting the presence of line and curve segments in images is the Hough transform. If a line can be mathematically defined by the equation $\mathbf{y = mx + b}$, then the infinite set of points that comprise this line all possess the same value of slope m and intercept b. The Hough transform uses this fact to map collinear or nearly collinear points in the (x,y) plane of the image space or feature space to the (m,b) coordinate of parameter space. One obvious discrepancy occurs when the line assumes a vertical orientation resulting in the slope approaching an infinite magnitude. This singularity can be avoided by using the normal parameterization of a line first suggested by Duda and Hart [Ref. 6]. The equation then appears in the following form:

$$\rho = x\cos\theta + y\sin\theta\ , \qquad 0 \le \theta \le \pi. \qquad (3.1)$$

Using the parametric equation of a straight line, each collinear (x,y) coordinate will map to a sinusoid in parameter space. After transformation, an infinite number of sinusoids are generated, each intersecting at the point in parameter space corresponding to the slope and intercept of the line.

1. Hough Transform Algorithm

The Hough transform adheres to the algorithm displayed in Table 2 [Ref. 7].

Table 2. HOUGH TRANSFORM ALGORITHM

```
For each $(x,y) \varepsilon P$ do
    For $\theta = 0, \pi, \Delta\theta$ do
        $\rho = x\cos\theta + y\sin\theta$
        $H(\theta,\rho) = H(\theta,\rho) + 1$
    end do
end do
```

2. Properties and Illustration

Using the normal parameterization of a line, the Hough transform method yields four main properties [Ref. 8]:

- A point in the feature space corresponds to a sinusoidal curve in the parameter space.
- A point in the parameter space corresponds to a line in the feature space.
- Points lying along a line in the feature space correspond to sinusoidal curves through a common point in the parameter space.

- Points lying along a sinusoidal curve in the parameter space correspond to lines through a common point in the feature space.

These properties are demonstrated using the illustration shown in Figures 12 and 13.

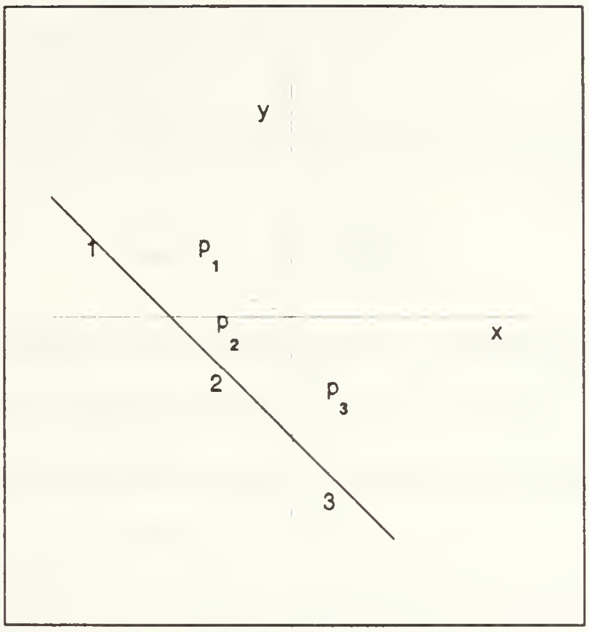

Figure 12. Test Image for the Hough Transform

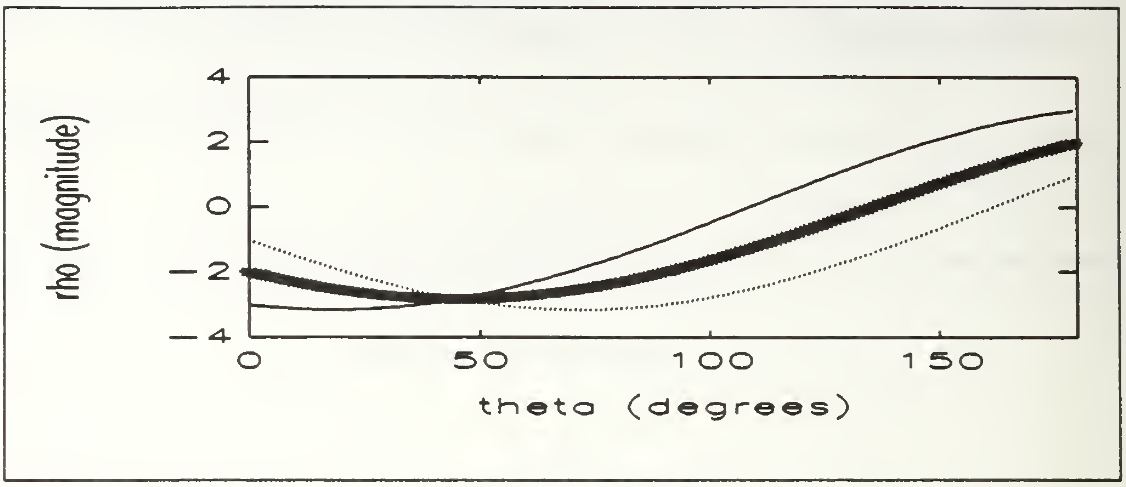

Figure 13. Sinusoids in Parameter Space from Collinear Points

The line segment, displayed in Figure 12, is used as a test image. The three indicated collinear points are extracted and used to generate the three sinusoids displayed in Figure 13. The intersection of the these three sinusoids in parameter space demonstrates the attractiveness of the Hough transform in locating collinear points in feature space. One only has to determine those points in parameter space where a large number of intersections occur to locate lines in feature space. The parameter space is viewed as a two dimensional array constructed by quantizing ρ and θ into cells. Each cell is assigned an accumulated value. Equation (3.1) maps each collinear point to a sinusoid in parameter space. Cells lying along the resultant sinusoid are incremented by one.

After the image has been transformed, accumulator cells having high counts indicate intersecting curves in parameter space and therefore lines in feature space [Ref. 7].

3. Reconstruction

After transformation, the accumulator array is evaluated to locate those cells possessing high counts. The problem of reconstructing the cluster center (ρ,θ) is solved by performing the inverse of equation (3.1). Solving equation (3.1) for x yields

$$x = \rho - y\left(\frac{\sin\theta}{\cos\theta}\right). \tag{3.2}$$

In order to implement this algorithm in a computer program, a reference point located at the center of the image is needed. Using a reference point simply introduces an offset (x_o, y_o) into equation (3.1):

$$\rho = (x-x_o)\cos\theta + (y-y_o)\sin\theta. \tag{3.3}$$

Solving this equation for x yields

$$x = x_o + \frac{\rho - (y-y_o)\sin\theta}{\cos\theta}. \tag{3.4}$$

B. IMPLEMENTATION

The Hough transform is implemented using HOUGH.FOR, a FORTRAN program written to take advantage of the points mentioned in Section A.2.

Previous programs of the Hough transform in [Ref. 8] and in [Ref. 2] involve the detection of signals present in noise and are therefore gray scale oriented. Because the output of GTT is binary, a new program, HOUGH.FOR, is used which is provided in Appendix B. The flow chart in Figure 14 depicts the flow of operation from output of GTT to tonal reconstruction.

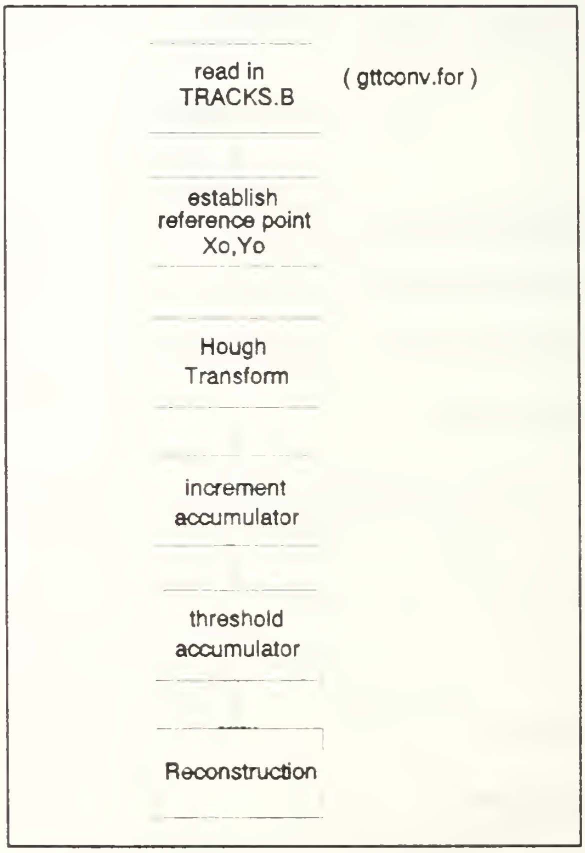

Figure 14. Hough Transform Flow Chart

1. **Source Code**

Because the output of GTT is a fixed length 128x512 byte file, a pre-filter is needed to convert TRACKS.B into a 256x256 byte array. The FORTRAN program, GTTCONV.FOR, that performs this operation is listed in Table 3.

Table 3. GTTCONV.FOR LISTING

```
      program goutconv
c     This program takes a fixed length 128x512 byte
c     output file from the GTT algorithm, which is
c     in binary format, and converts it into a 256x256
c     byte array. The output file, OUTFILE.DAT, is
c     now suitable for input into the Hough Transform
c     algorithm.
      byte b1_image(128,512),b2_image(256,256)
      integer i_image(256,256)
      open(unit=1,name='tracks.b',status='old',
     *     access='direct',recordsize=128,maxrec=128)
      open(unit=2,file='outfile.dat',status='new',
     *     access='direct',recordsize=64,maxrec=256)

      do i=1,128
         read(1'i)(b1_image(i,j),j=1,512)
         do j=1,256
            b2_image((i*2)-1,j)=b1_image(i,j)
         end do

         do j=257,512
            b2_image(i*2,j-256)=b1_image(i,j)
         end do
      end do

      do i=1,256
         write(2'i)(b2_image(i,j),j=1,256)
      end do
      close(unit=1)
      close(unit=2)
      end
```

Subroutine h in HOUGH.FOR performs the Hough transform. θ is incremented from 0 to π radians in intervals of 1/256 radians. For each nonzero pixel in the image, the Hough transform equation is executed and each cell in the accumulator array that is traversed by the resulting sinusoid is incremented by one. After performing the Hough transform, the accumulator array is thresholded and normalized for display on the IM-4000 Image Manager image processing system. This image processor requires that the scale for normalization range be from 0 to 243. Levels 244 to 255 are internally reserved [Ref. 9]. Cells in the accumulator array greater than or equal to a user defined threshold will map to the most likely set of collinear points in feature space. Subroutine R in HOUGH.FOR is then called to reconstruct lines from cluster points in the accumulator array. After determining the number of cluster points and their location, (ρ,θ), the reconstruction equation (3.4) is used.

2. Output

a. Test.dat

The test image, TEST.DAT, shown in Figure 15, is used as the input file for the Hough transform. The result of this transform is shown in Figure 16.

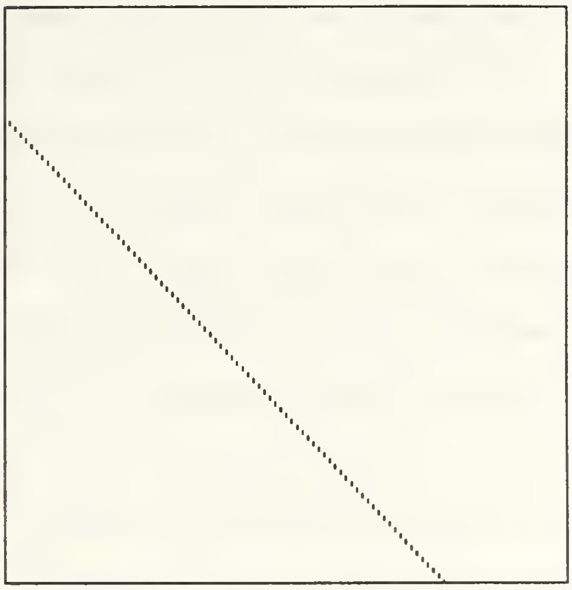

Figure 15. Test Image in
Space Domain,
TEST.DAT

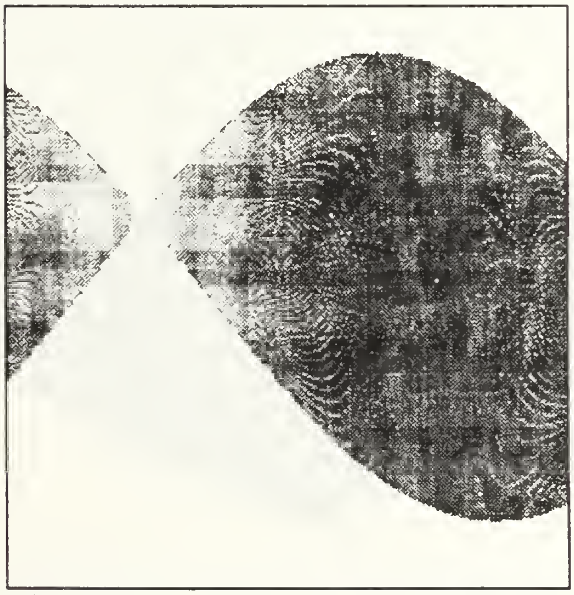

Figure 16. Hough Transform of
TEST.DAT

As predicted, the sinusoids intersect at a common (ρ,θ) location. The reconstruction of this Hough transformed image is displayed in Figure 17, which demonstrates that in the absence of noise the Hough transform can faithfully and easily reproduce the line test image. The accumulator array for the Hough transform contains one cell that possesses the highest number of sinusoids traversing it, and therefore the highest count. In three dimensions, this appears as a lone peak rising from a plane defined by ρ and θ. The 3-D plot of the Hough transform in Figure 16 is shown in Figure 18.

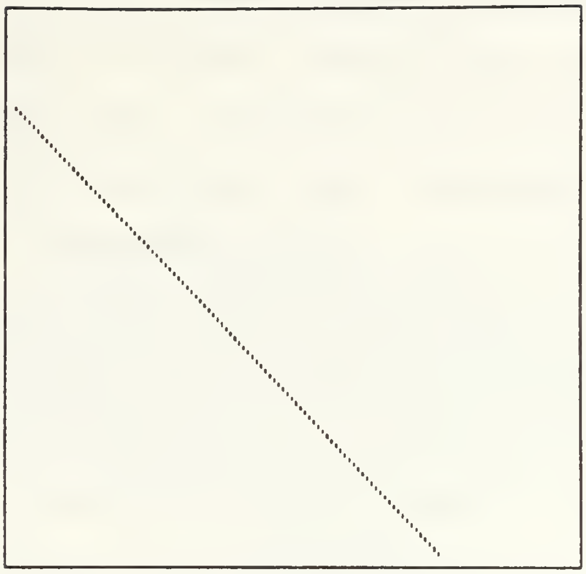
Figure 17. Reconstructed Image of TEST.DAT

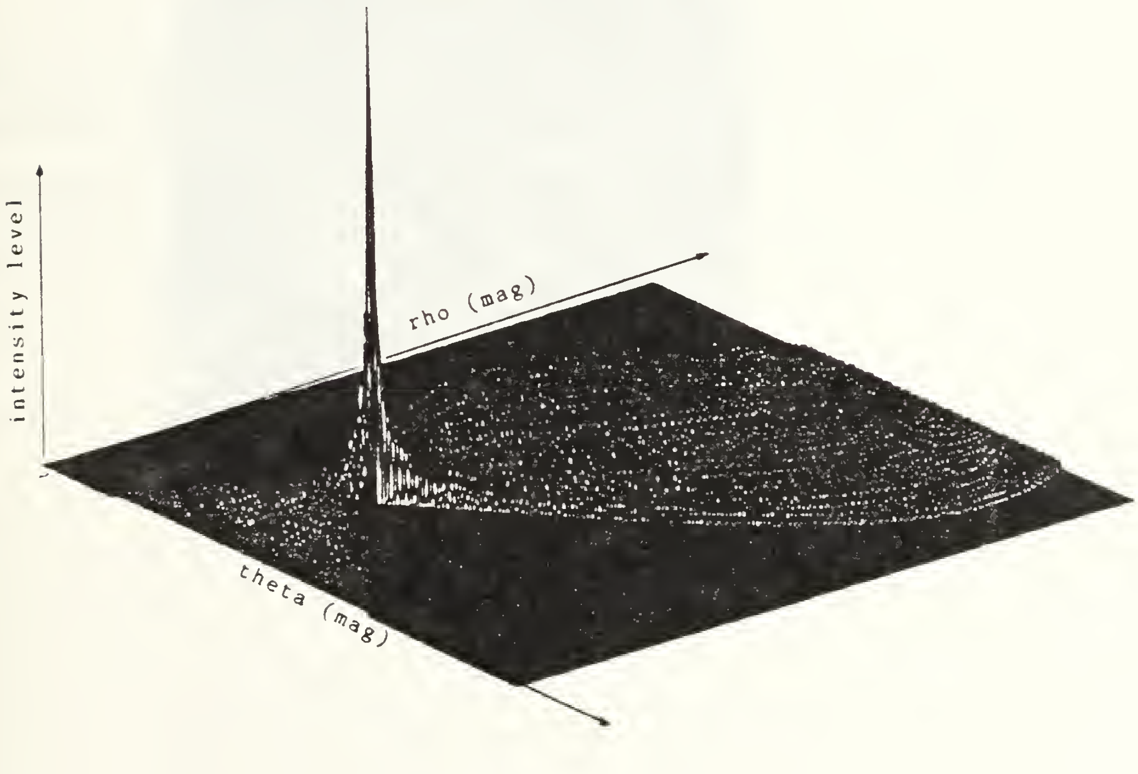

Figure 18. 3-D Plot of the Hough Transform of TEST.DAT

b. Tracks.b (Outfile.dat)

The output of GTT, TRACKS.B, shown in Figure 7, is converted, as discussed previously, into OUTFILE.DAT and used as input for HOUGH.FOR. The Hough transform of this file is displayed in Figure 19. Because TRACKS.B contains numerous false alarms in the form of small line segments, the Hough transform appears to be extremely noisy and convoluted. After the accumulator array for this transform is thresholded at a value of 100%, reconstruction produces the single most dominant tonal in the LOFARGRAM, which is shown in Figure 20.

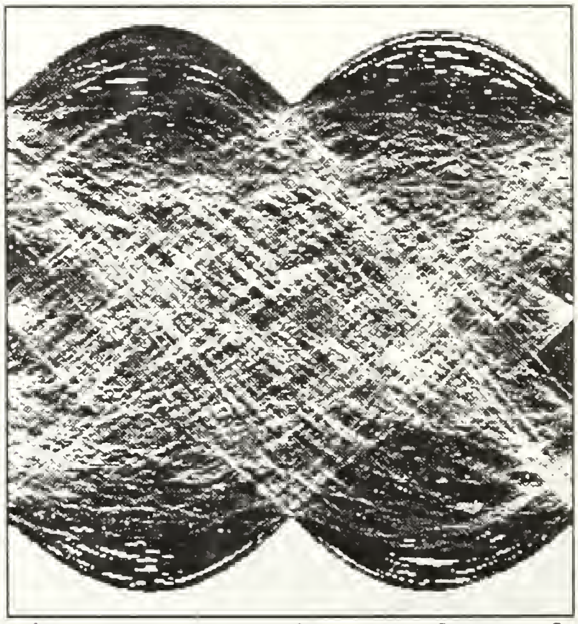

Figure 19. Hough Transform of GTT Results, TRACKS.B

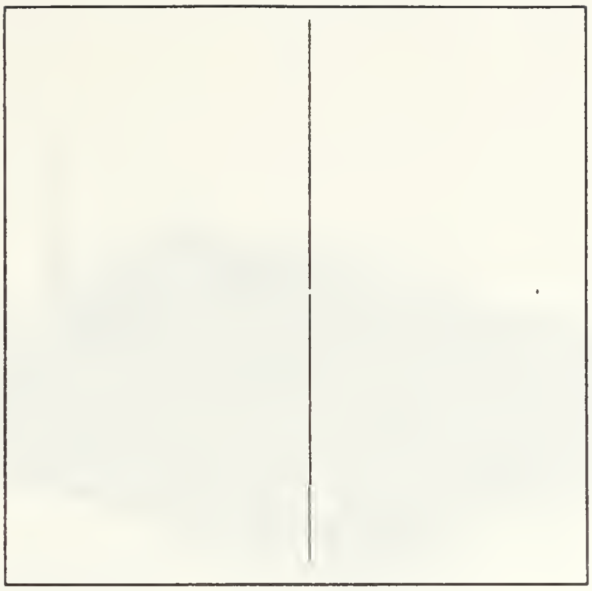
Figure 20. Reconstructed Dominant Tonal, with 100% Threshold, of Figure 19

The reconstruction subroutine in HOUGH.FOR did not sense the less evident tonals because the accumulator cells associated with those lines contain counts much lower than the cell linked to the most prevalent tonal. This is evident in Figure 21 which shows the 3-D plot of Figure 19.

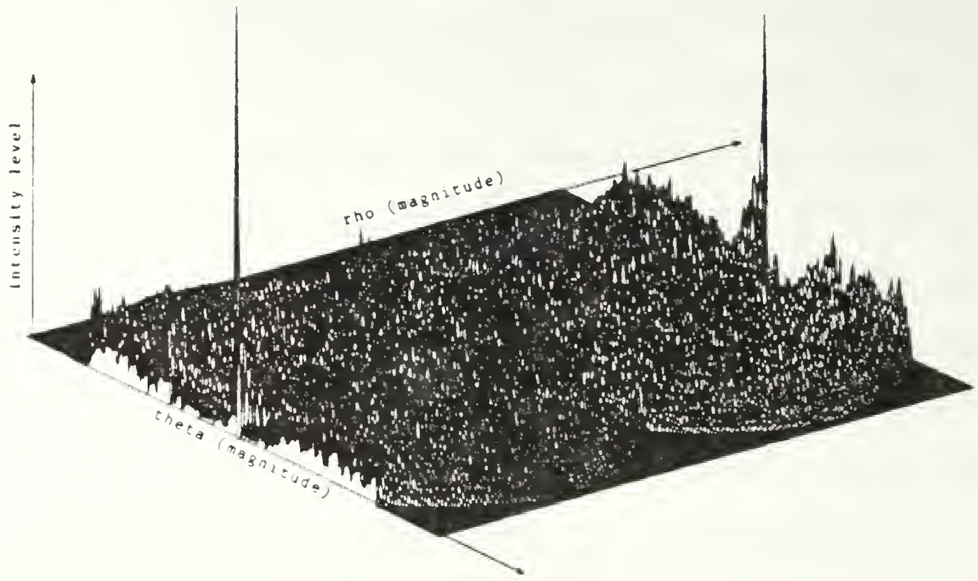

Figure 21. 3-D Plot of the Hough Transform of GTT Results, TRACKS.B

Even with a threshold value of 70%, the most dominant tonal is still the only one reconstructed, as shown in Figure 22. Clearly then, an alternative is required to extract those cluster centers associated with tonals of interest in the LOFARGRAM.

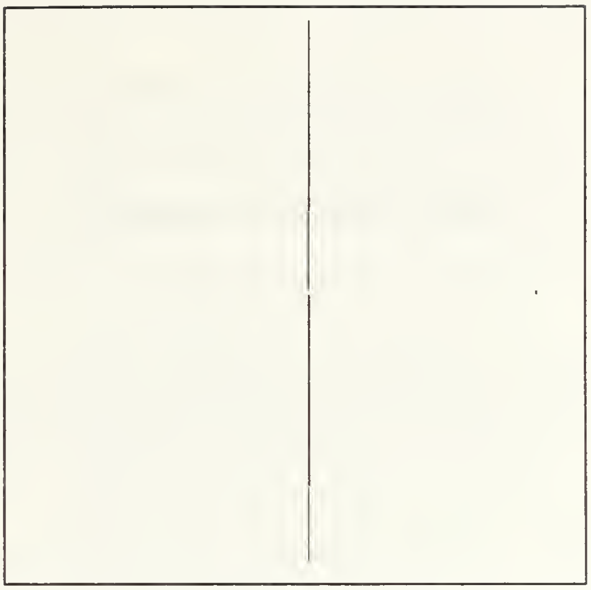
Figure 22. Reconstructed Dominant Tonal, with 70% Threshold, of Figure 19

The issue is how to find relevant cluster centers in the parameter space of the Hough transform. In the next chapter, a heuristic (greedy) type of algorithm is developed to locate cluster centers.

IV. HEURISTIC SEARCH

A. BACKGROUND

Clearly a simple thresholding operation is insufficient to reconstruct all applicable tonals from the accumulator array. A method needs to be developed to effectively sort the accumulator array and reconstruct lines from accumulator cells of interest. Two methods, LAS cluster technique and sorting technique, have been previously studied as a means to perform cluster analysis [Ref. 2]. The Land Analysis System (LAS) is a software package encompassing a variety of functions designed to process and analyze image data. Based on a Digital Equipment Corporation (DEC) VAX 11/780 computer, the LAS runs under the VMS operating system [Ref. 10]. Three programs of interest within the multispectral processing functions of LAS include HINDU, ISOCLASS, and KMEANS. The HINDU program performs an unsupervised classification based on multidimensional histograms. ISOCLASS performs an unsupervised cluster classification and KMEAN groups input image pixel values into a predetermined number of clusters. In addition to applying a sort based on threshold, Wang studied these three programs as an alternative cluster analysis method. This thesis examines a possible third procedure, heuristic search.

B. THEORY

The heuristic search method involves scanning the accumulator array and determining the location of groups or clusters of cells or points which possess values equal to or greater than a user defined threshold. The threshold is based on percentage with 100% assigned to the cell with the highest count. As the search proceeds, the clusters are condensed into a specific cluster of cells which will then be reconstructed. In order to implement this procedure, a cost function is defined.

1. Cost Function

The total cost function is formed from the addition of a cost function based on the number of points ($cost_N$) and a cost function based on distances between points ($cost_D$). Symbolically, these functions are illustrated below.

$$cost_N = K_n N \tag{4.1}$$

$$cost_D = K_d D = \sum_{u_i} \sum_{x_j} |u_i - x_j| \tag{4.2}$$

$$totalcost = cost_N + cost_D \tag{4.3}$$

Where $\mathbf{K_n}$ and $\mathbf{K_d}$ are scalar constants used to weight $\mathbf{cost_N}$ and $\mathbf{cost_D}$, respectively, $\mathbf{u_i}$ represents cluster group **i** and $\mathbf{x_j}$ represents the **j** points within cluster group **i**. An illustration shown in Figures 23 through 26 is used to demonstrate the cost function.

Figure 23 shows an example of an accumulator array with cells containing values at or above the threshold depicted as black points on the grid. Numbers listed vertically mark rows in the array; horizontally listed numbers mark columns. The black x's indicate cluster groups.

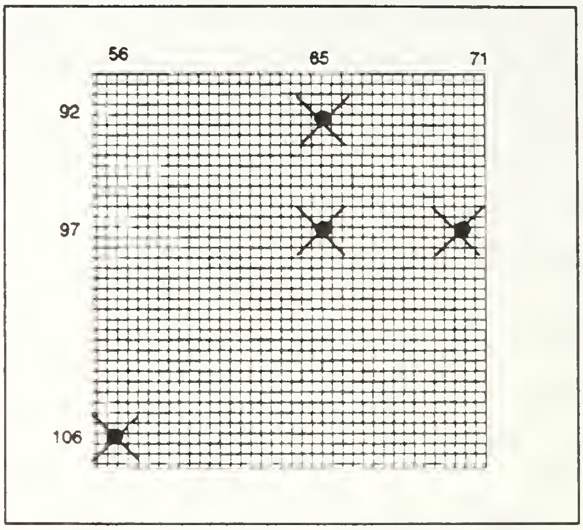

Figure 23. Test Image of Heuristic Search with N=4 D=0.0

At this point in the calculations, N equals four and a marker is placed on each point yielding a distance from each marker to its respective point zero. The minimum distance between markers is now computed, and a new marker is placed at the mid-point between the two closest markers. In Figure 24, N now equals three, and the distance cost associated with the new marker is calculated. The distance used in this calculation is the distance from the new marker to each of the original cluster markers. At the completion of each state, the total cost is computed and tabulated.

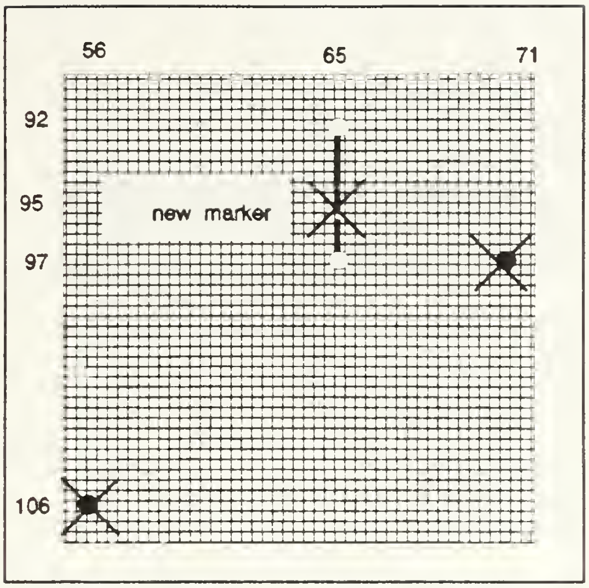

Figure 24. Test Image of
Heuristic Search
with N=3 D=5.0

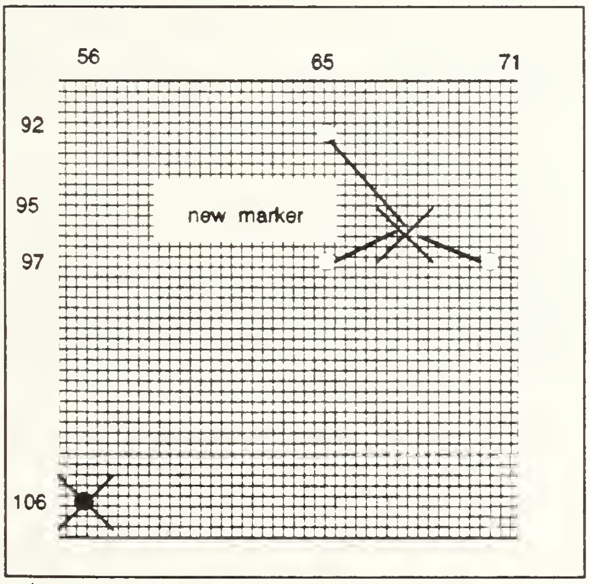

Figure 25. Test Image of
Heuristic Search
with N=2 D=11.32

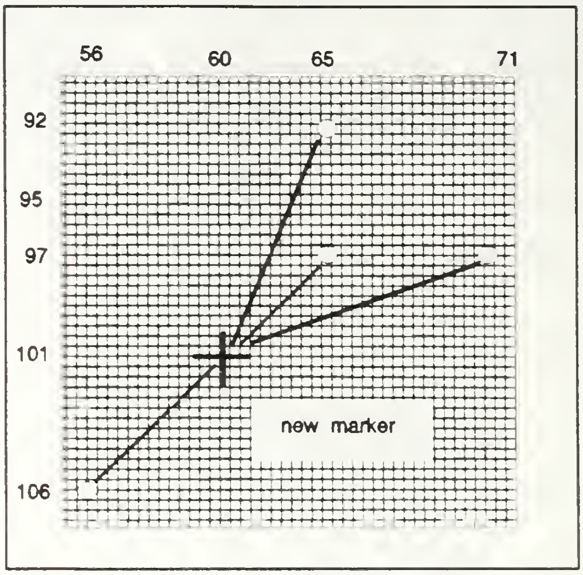

Figure 26. Test Image of Heuristic Search with N=1 D=34.81

When all states have been evaluated, the state possessing the minimum overall cost is reconstructed back into line tonals in feature space. The three cost functions are shown in Figures 27, 28, and 29. The total cost function for the previous example is plotted against states one through four.

It is evident that state two representing the two cluster group situation is the lowest cost configuration which would be the answer for the number of reconstructed clusters.

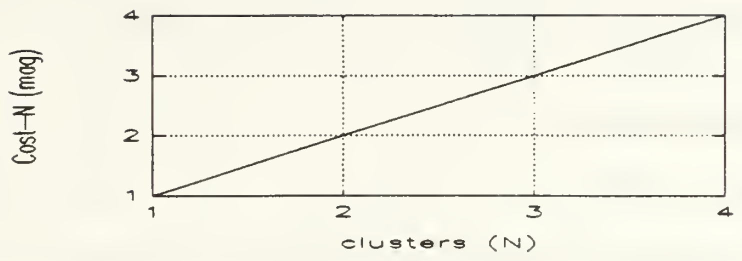

Figure 27. Cost_N

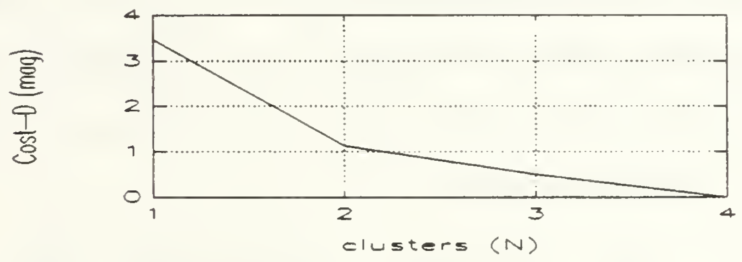

Figure 28. Cost_D

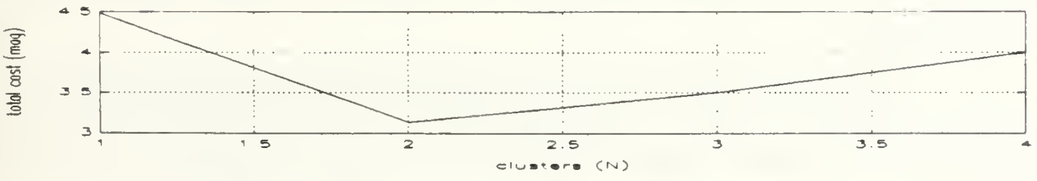

Figure 29. Total Cost

After experimentation, scalar constants $\mathbf{K}_n$=1.0 and $\mathbf{K}_d$=0.1 were chosen to ensure that, within the total cost function, $\mathbf{cost}_N$ is properly offset by $\mathbf{cost}_D$.

C. IMPLEMENTATION

1. Source Code

The Heuristic Search method, written in the FORTRAN programming language, is provided in Appendix C as HS.FOR. This program takes the 256x256 accumulator array constructed in HOUGH.FOR as input and determines the minimum cost configuration. This configuration is then provided as input for RECONST.FOR which reconstructs the optimal minimum cost configured accumulator array back into lines in feature space.

a. HS.FOR Program

It is important to remember that the image supplied to HS.FOR has already been thresholded by HOUGH.FOR. Pixels containing values equal to greater than the threshold are retained and those below the threshold are set to zero (pixel value 0 equates to the color black for display purposes).

After conversion from a 256x256 byte image into an integer format, the x,y locations of the N non-zero pixels are stored in four 1xN arrays: S_PEAK_ROW, S_PEAK_COL, C_PEAK_ROW, and C_PEAK_COL.

S_PEAK_ROW and S_PEAK_COL store values from the input sample image and remain unchanged as the reference image. C_PEAK_ROW and C_PEAK_COL represent the cluster image and store the locations of the current cluster groups. The length of c_peak_row and c_peak_col arrays will change as the program progresses from state n=N non-zero pixel or peaks to state n=1. Initially, these four arrays are identical.

Within the main program loop, subroutine CENTROID performs the majority of the processing on the cluster image. Sequentially, this subroutine determines the minimum distance between peaks and places a new marker at the mid-point of the two peaks involved in the minimum distance calculation. It adjusts c_peak_row and c_peak_col arrays to include the newly calculated midpoint and determines the distance factor to be used in the distance cost portion of the total cost function. It then performs the next set of distance calculations as the loop is executed again. After all states have been evaluated, subroutine min_cost is called to determine and output the cluster group configuration representing the minimum total cost.

b. RECONST.FOR Program

This program takes the output from HS.FOR as input and performs reconstruction calculations nearly identical to those found in HOUGH.FOR.

The main difference is that the pixels being reconstructed now represent the cluster configuration of the minimum cost.

2. Testing

a. Threshold 30%

At a threshold of 30%, ten cells in the accumulator array are encountered and are listed in Table 4.

Table 4. ACCUMULATOR ARRAY CELLS OF HOUGH TRANSFORM PARAMETER SPACE

Cells (Peaks)	Row	Column
2	256	127
3	253	128
4	254	128
5	255	128
6	256	128
7	1	129
8	2	129
9	3	129
10	4	130

After reconstruction from parameter space to feature space, these ten peaks yield ten overlapping lines shown in Figure 30, depicting the most dominant tonal in the original LOFARGRAM.

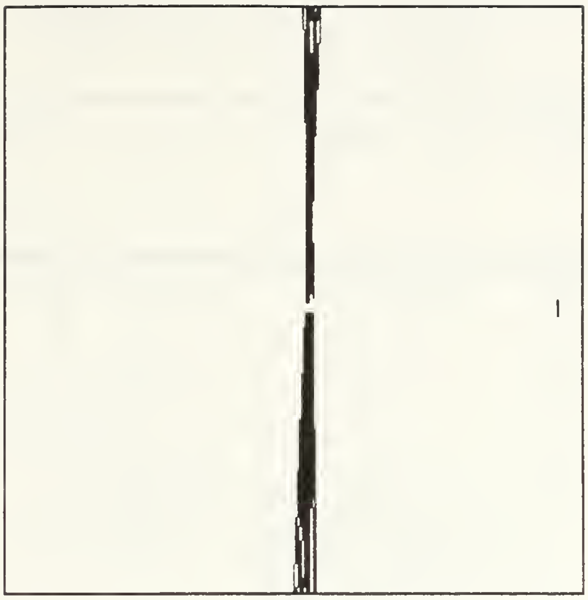
Figure 30. Reconstructed Dominant Tonal with 30% Threshold of Figure 19

In order to implement the heuristic search cluster analysis, the image shown in Figure 30 is used as the input image for HS.FOR. The ten lines correspond to ten states initially. As the program proceeds from state n=10 to state n=1, $cost_N$, $cost_D$, and total cost are computed. Cost function values for this example are listed in Table 5.

Table 5. COST FUNCTION VALUES FROM FIGURES 27, 28, AND 29

state, n	cost_N	cost_D	total cost
10	10.0	0.0000	10.0000
9	6.0	1.0000	9.0000
8	6.0	2.0000	8.2000
7	7.0	3.0000	7.3000
6	6.0	4.0000	6.4000
5	5.0	5.4142	5.5414
4	4.0	6.8284	4.6828
3	3.0	8.2426	3.8243
2	2.0	10.2426	3.0243
1	1.0	1265.051	127.051

It is evident that state n=2 represents the minimum cost configuration. Two peaks in the accumulator array result in two lines in feature space as shown in Figure 31.

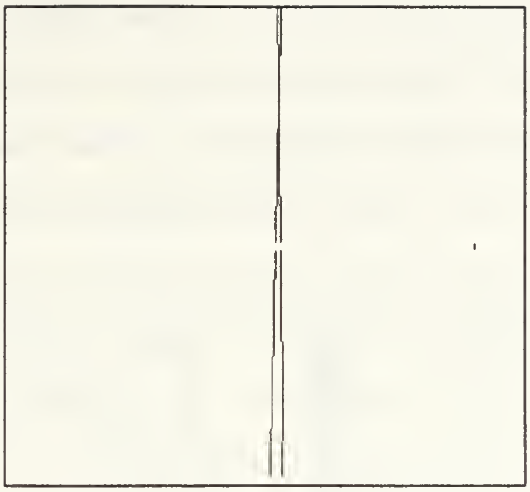
Figure 31. Reconstruction of Figure 30 after Heuristic Search

b. Threshold 20%

When the threshold is reduced to 20%, the number of accumulator array cells or peaks meeting the threshold criteria is increased to 32. The 32 lines reconstructed from these pixels without heuristic search are shown in Figure 32.

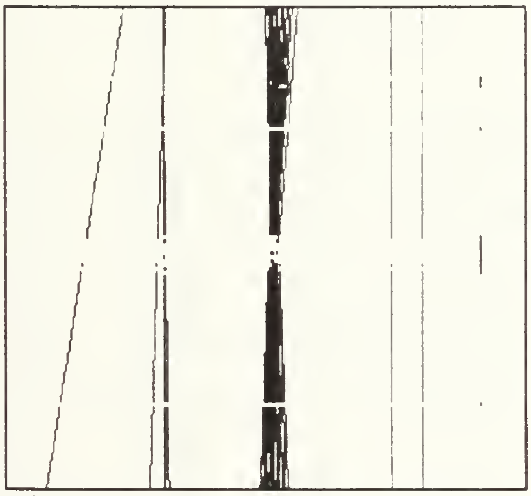
Figure 32. Reconstructed Tonals with 20% Threshold of Figure 19

The sloping line to the far left in the image represents the top portion of the slowly varying curved tonal in the original LOFARGRAM. The next set of vertical lines depicts the less evident tonals while the majority of lines are centered around the position of the dominant tonal. The two vertical lines to the far right are spurious and represent noise pixels that met the threshold criteria in the accumulator array.

The heuristic search yields a minimum cost state of eight cells, and the reconstruction of which is shown in Figure 33. The large number of lines near the dominant tonal has been reduced to three. It is interesting to note that the two spurious lines associated with noise are retained.

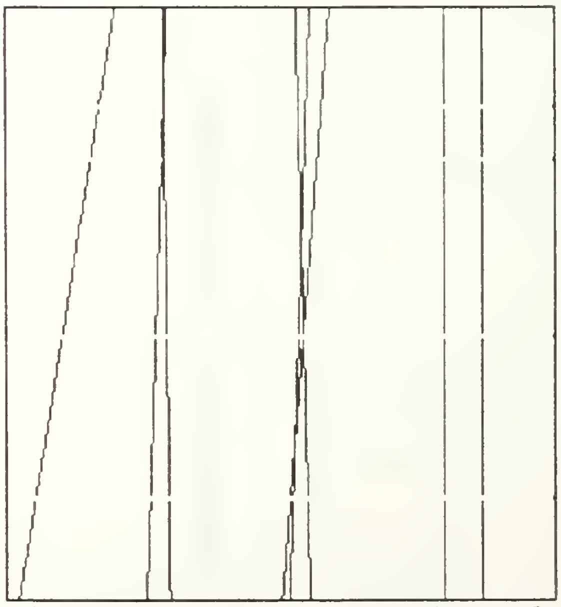

Figure 33. Reconstruction of Figure 32 after Heuristic Search

c. Threshold 19%

Lowering the threshold to 19% yields 40 peaks in the accumulator array corresponding to the 40 lines shown in the image in Figure 34.

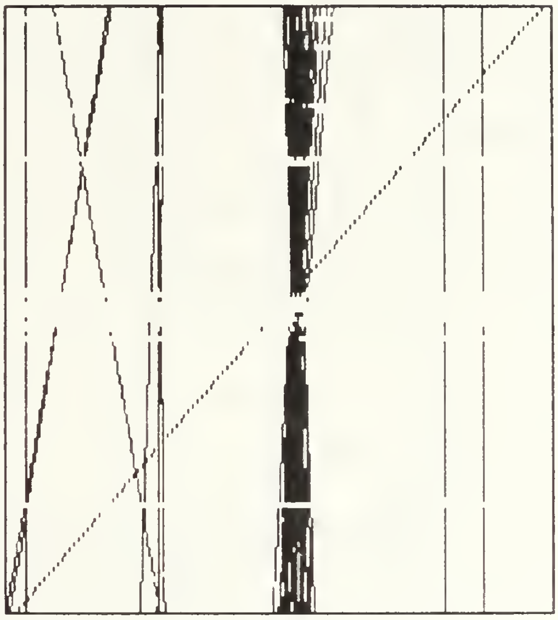

Figure 34. Reconstructed Tonals with 19% Threshold of Figure 19

New additions from the 20% threshold image include the vertical line to the far left representing another spurious noise cell in the accumulator array. It also includes two more sloping lines depicting the mid and lower portions of the swept tonal from BEAM00.BIN.

Eleven cells represent the minimum cost state in the accumulator array resulting in 11 lines reconstructed in feature space as shown in Figure 35.

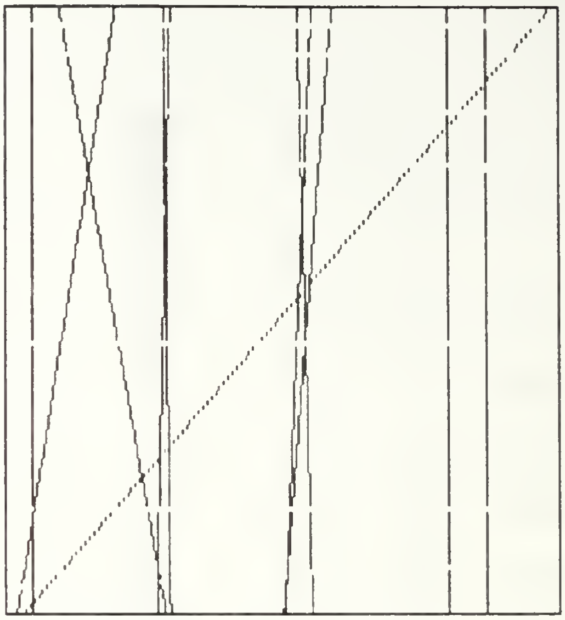
Figure 35. Reconstruction of Figure 34 after Heuristic Search

The spurious noise lines have unfortunately been preserved while the majority of lines clustered around the dominant tonal position have been reduced to three. It is evident that the heuristic search cluster analysis, although working well, needs to be improved.

3. Improvements

In order to reduce the effects of noise while retaining those cells in the accumulator array associated with the signal tonals, the input image for the Hough transform is subjected to further processing. MULTIPLY.FOR, provided in Appendix D, is used to multiply the output of GTT, TRACKS.B, with the original LOFARGRAM, BEAM00.BIN. The resultant image is a replica of TRACKS.B with the exeception that every non-zero pixel contains the value of BEAM00.BIN at that (i,j) location.This image now emphasizes the signal tonals. The output of MULTIPLY.FOR, OUTIMG.DAT, is now used as the input image for the Hough transform. Because HOUGH.FOR is binary image oriented, changes are made in the source code of this program to accomodate the 0-255 gray scale orientation of OUTIMG.DAT. These changes are listed as HOUGH_G.FOR in Appendix E.

a. Threshold 17%

At a threshold of 17%, 38 peaks in the accumulator array meet the threshold criteria. When this array is fed into the heuristic search algorithm, HS.FOR, nine peaks are retained for reconstruction. The output of HOUGH_G.FOR is shown in Figure 36; the output of HS.FOR is depicted in Figure 37.

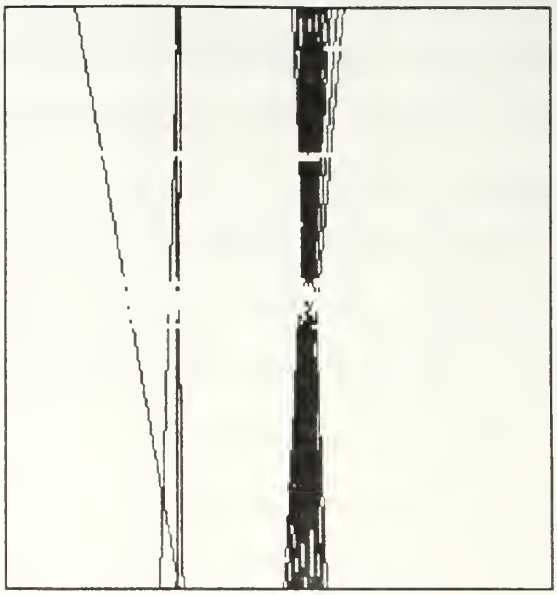
Figure 36. Reconstructed Tonals with 17% Threshold of Figure 19

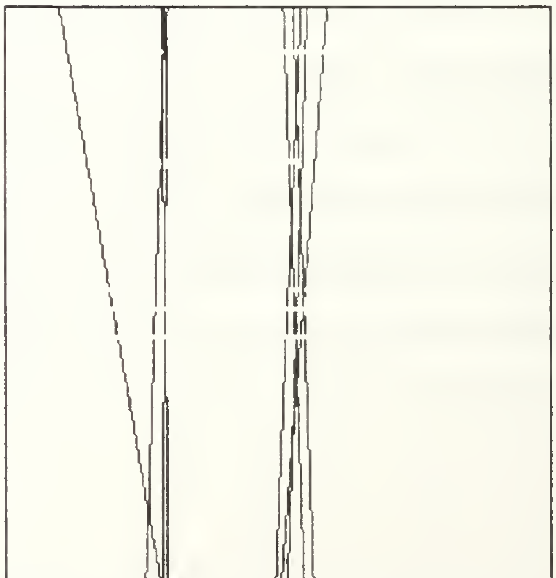
Figure 37. Reconstruction of Figure 36 after Heuristic Search

Figure 37 shows that the manipulation of the input image by the multiplication of TRACKS.B and BEAM00.BIN results in an output image from heuristic search where the main, secondary, and portions of the swept tonal are reconstructed without the introduction of noise. The threshold is then reduced to 16% resulting in 52 peaks meeting the threshold criteria. After the heuristic search is completed, 11 peaks are retained which reconstruct 11 lines in feature space. These images are shown in Figures 38 and 39.

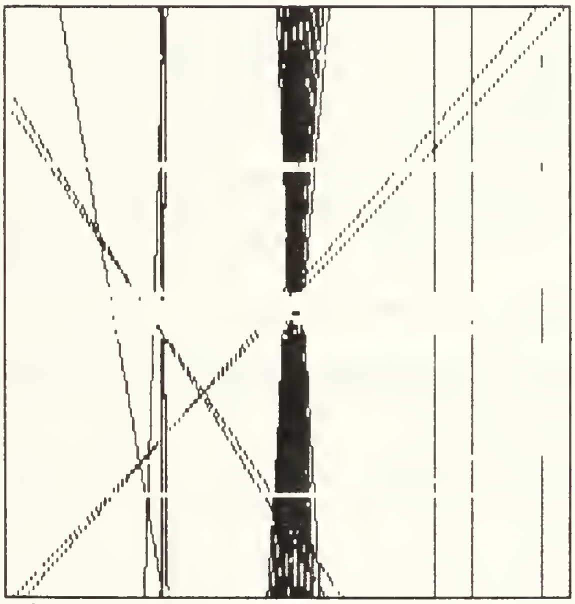

Figure 38. Reconstructed Tonals with 16% Threshold of Figure 19

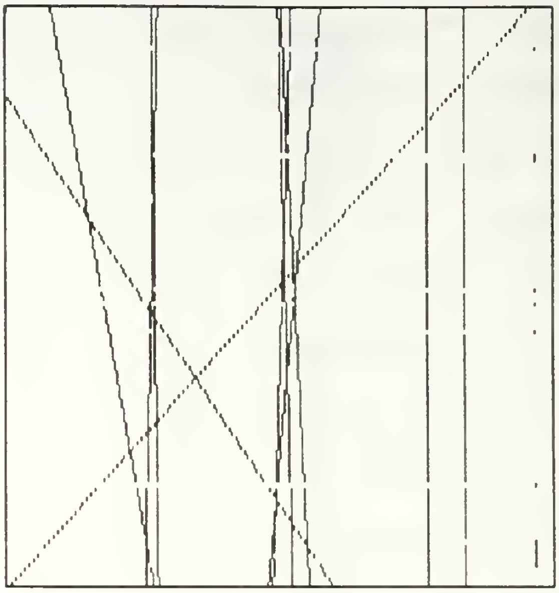
Figure 39. Reconstruction of Figure 38 after Heuristic Search

Again the main, secondary, and swept tonals are retained. However, the two vertical lines to the far right in Figure 39 are spurious resulting from two noise cells in the accumulator array passing the threshold criteria. With the proposed manipulation, it is obvious that more line fragments of desired tonals can be extracted out of the original image.

V. CONCLUSIONS AND RECOMMENDATIONS

A. GENERAL

This thesis has presented an examination of the combination of three algorithms: GTT, Hough Transform, and Heuristic Search to enhance the detection of spectral tracks of underwater objects. Because signals of interest are best visualized using LOFARGRAMs, LOFARGRAM analysis remains an integral part of tracking in the frequency domain. The combined approach of the application of these algorithms is used to take full advantage of their respective characteristics. Conclusions drawn from this research are listed below for each of the three algorithms.

1. GTT Algorithm

- Because the output is already in the form of potential tracks, it can be used for further processing.
- This algorithm is able to distinguish between connected signals and relatively poorly connected noise.
- GTT can process multiple stable tonals clearly.

2. Hough Transform Algorithm

- This transform can extract desired line fragments of stable and sweeping tonals from planar point sets.
- The Hough transform can handle spectral tracks buried in heavy background noise.

- This algorithm can operate efficiently on both binary and gray scale images.

3. Heuristic Search Algorithm

- Heuristic Search provides an easily understood alternative to LAS routines and sorting technique for cluster analysis.
- This algorithm provides a search foundation for further image processing techniques such as simulated annealing.

B. RECOMMENDATIONS

Several unresolved problems remain at the conclusion of this thesis. The cost function used in determining which cluster(s) to reconstruct in the heuristic search algorithm needs refinement. Presently, only the number of peaks and distances between peaks are considered. One possible improvement might include the addition of a height function which would favor those cells in the accumulator array with higher counts. This could offset the effects of noise and emphasize the desired tonals in the LOFARGRAM. A second recommendation involves expanding heuristic search into the simulated annealing algorithm. Simulated annealing is very effective in determining the global optimal solution and could be used in cluster analysis by finding the optimal minimum solution to the cost function.

To operate effectively, the simulated annealing algorithm needs to be given an initial guess of the location of potential tracks. The fragmented output from GTT is not suitable as an input state for simulated annealing. By using the Hough transform to convert the discontinuous points in GTT's output to line tonals, an initial guess can be provided for simulated annealing which is close to the actual position of the tonals in the original LOFARGRAM. Research has been conducted to apply the technique of simulated annealing to sonar track detection, but it only involved the use of laboratory generated synthetic data sets [Ref. 11].

APPENDIX A: GRAPH THEORETIC TRACKER SOURCE CODE

MAIN.C

```
#define MAIN
#include <stdio.h>
#include <time.h>
#include <signal.h>
#include "part.h"

main( argc, argv )
int argc;
char *argv[];
{
        int     line=0, i;
        long    starttime, stoptime;
        FILE    *initialise(), *infile;
        VERTEX  lastnode, GenerateNodes();
        void    writepart(), partition(), update(),
                DumpNodes(), DumpTable();

        /********************/
        /* Body of Program */
        /********************/
        (void) printf ("%s\n\n", verid);
        infile = initialise( argc, argv );
        (void) printf ("Building tree ...\n");
        lastnode = GenerateNodes();
        (void) printf("Nodes=%d\n\n", lastnode);

        for(;;) {
                line++;
                update( lastnode, infile );
                if(feof(infile)) break;

                (void) printf ("Processing group %3d",line);
                (void) time(&starttime);
                partition(lastnode);
                (void) time(&stoptime);
                stoptime -= starttime;
                (void)
printf("(%2ld:%2ld)\n",stoptime/60,stoptime % 60);
                writepart( lastnode-1 );
        }
}
```

PART.H

```
#define verid           "*** Partimg v5.0 ***"
#define MAXNODE         2500          /* max. tree nodes */
#define MAXPIXELS       256           /* pixels per line */
#define MAXLINES        3             /* number of lines */
#define nullset         -1
#define TRUE            1
#define FALSE           0
#define BIGINT          10000
typedef char            BOOLEAN;
typedef short           VERTEX;       /* node in tree */

/* typedef int   void;     */   /* dummy void type if
compiler lacking one */


/* Define Global Variables and Arrays */
#ifdef MAIN
short int       setsize[MAXNODE][MAXLINES];
short int       setsize1[MAXNODE][MAXLINES];
unsigned int        size[MAXNODE];/* tree node sizes */
unsigned int        acc[MAXNODE];/* cost funct track acc */
int         table[MAXNODE];
VERTEX          opt[MAXNODE];
int             KLimit, lines, npix;/* Case Size */


#else
extern short int    setsize[][MAXLINES];
extern short int    setsize1[][MAXLINES];
extern unsigned int size[];/* node sizes      */
extern unsigned int acc[];/* c.f. track acc. */
extern int      table[];
extern VERTEX   opt[];

extern int      KLimit, lines, npix;
#endif
```

INITIAL.C

```
#include <stdio.h>
#include <string.h>
#include "part.h"

/* Parse Input Arguments */
FILE *initialise ( argc, argv )
int argc;
char *argv[];
{
    char    imagefile[64], *strcpy();
    FILE    *infile;
    void    exit(), usage(), valerr(), err();


    if(argc>1)
      if(strcmp(argv[1],"/h")==0||argc>5) usage(argv[0]);


        switch( argc )
        {
           case 5: KLimit = atoi( argv[4] );
           case 4:  lines = atoi ( argv[3] );
           case 3:   npix = atoi( argv[2] );
           case 2: (void) strcpy( imagefile, argv[1] );
           default: ;
        }

        switch( argc )
        {
           case 1: (void) fprintf(stderr,"Input
filename:\t\t" );
           (void) scanf("%s", imagefile);
           case 2: (void) fprintf(stderr,"Line length:\t\t"
);
           (void) scanf("%d", &npix);
           case 3: (void) fprintf(stderr,"Segmentation
lines:\t");
           (void) scanf("%d", &lines);
           case 4: (void) fprintf(stderr,"Enter KLimit
value:\t");
           (void) scanf("%d", &KLimit);
        }
        infile = fopen(imagefile,"r+b");
        if (infile==NULL) err ("Unable to open input
                              file.");

        if (npix<1 || npix>MAXPIXELS)
           err("Invalid number of pixels");

        if (lines>MAXLINES || lines<1)
           err("Invalid number of lines");

        if (KLimit<1 || KLimit>4)
           err("Invalid KLimit");

        (void) unlink("tracks.b");
        return(infile);
}

void usage(argv)
char *argv;
{
    (void) printf("\tusage: %s [input] [pixels] [lines]
[KLimit]\n", argv);
}

void err (arg)
char *arg;
{
    (void) printf ("%s\007\n", arg);
}
```

GENNODE.C

```
#include "part.h"
int     node=0;

VERTEX  GenerateNodes ()
{
    register int  i, j;
    void Gen();

    /* Initialise setsize */
    for (i=0;i<MAXNODE;i++)
        for (j=0;j<lines;j++)
            setsize1[i][j] = setsize[i][j] = nullset;

    /* Generate tracks */
    for(i=1;i<npix;i++)
    {
        setsize[node][0] = i;
        Gen(0, i);
    }

    /* Fill in node tables */
    for (i=0;i<lines;i++)
           for (j=1;j<node;j++)
              if( setsize[j][i] == nullset)
                setsize[j][i] = setsize[j-1][i];

    for (i=0;i<lines;i++)
           for (j=0;j<node;j++)
              setsize1[j][i]=setsize[j][i]+1;

    for (i=0;i<node;i++)
           size[i]=0;
    for (i=0;i<node;i++)
           for (j=0;j<lines;j++)
              size[i] = setsize[i][j] + size[i];

    return(node);
}
```

```
void Gen (LineNumber,BinNumber)
int LineNumber, BinNumber;
{
    register int    i, j;
    void            err();

    if (node >= MAXNODE) err ("Too many nodes!");

    LineNumber++;
    for (i = -KLimit ;i<=KLimit;i++)
    {
        j = BinNumber+i;
        if (j > 0 && j < npix)
        {
            setsize[node][LineNumber] = j;

            if (LineNumber == lines-1)
                node++;
            else
                Gen(LineNumber, j);
        }
    }
}
```

UPDATE.C

```
#include <stdio.h>
#include "part.h"

void update(lastnode,infile)
int lastnode;
FILE *infile;
{
    register int    i, j;
    unsigned char   buffer[MAXPIXELS];
    int     weight[MAXLINES][MAXPIXELS];

    for( j=0; j<lines; j++ )
    {
        if(fread(buffer,1,npix,infile)==0) return;
          for( i=0; i<npix; i++ )
             weight[j][i] = (int) buffer[i];
    }

    /* set up signal estimate acc */
    for (i=0;i<lastnode;i++)
           acc[i]=0;
    for (i=0;i<lastnode;i++)
        for (j=0;j<lines;j++)
            acc[i]+=weight[j][setsize[i][j]];

    /* set up table */
    for( j=0; j<lines; j++ )
       for( i=1; i<npix; i++ )
           weight[j][i]+=weight[j][i-1];

    for( i=0; i<lastnode; i++ )
        table[i]=1;
    for (i=0;i<lastnode;i++)
        for (j=0;j<lines;j++)
             table[i]+=weight[j][setsize[i][j]-1];
}
```

PARTIT.C

```
#include "part.h"
#include <stdio.h>
void partition( nodecount )
int nodecount;
{
    register int    i, j, c;
    BOOLEAN         status;
    int             cost[MAXNODE];
    FILE *fp,*fopen();
    int minval;costfunction();
    for (i=0;i<nodecount;i++)
    {
        cost[i] = BIGINT;
        opt[i]  = nullset;
    }
    for (i=1;i<nodecount;i++)
    {
           for( j=0; j<i; j++ )
           {
              status=TRUE;
              for(c=0;c<lines;c++)
                if(setsize[i][c]<=setsize1[j][c])
                {
                status=FALSE;
                break;
                }
                if (status)
                {
                c = costfunction( i, j ) + cost[j];
                if( (cost[i]) >= c )
                {
                    cost[i] = c;
                    opt[i] = j;
                }
                }
           }
    }
}
int costfunction( i, j )       /* cost of segment */
VERTEX i, j;
{
    register int icost;
    int COST;
    icost = (int) (table[i]-table[j])*6; /*threshold*/
    icost /= (int) (size[i]-size[j]);
    return(icost - (int) acc[i] );
}
```

LIBRARY.C

```
#include <stdio.h>
#include "part.h"

void writepart( node )      /* write partition */
VERTEX node;
{
    register int i, j;
    unsigned int bitmap[MAXLINES][MAXPIXELS];
    FILE *outfile;

    for( i=0; i<lines; i++ )
           for( j=0; j<npix; j++ )
               bitmap[i][j] = 0;   /* bitmap 0 is
                                      gray scale black */

    node = opt[node];
    while( node != nullset )
    {
        for( i=0; i<lines; i++ )
               /* pixel quantization for the output
                  image is 8 bits/pixel, with the
                  gray scale running from 0 (black)
                  to 255 (white) */
               bitmap[i][setsize[node][i]] = 255;
               node = opt[node];
    }

    outfile = fopen("tracks.b","a+b");
    for (i=0;i<lines;i++)
           (void) fwrite(bitmap[i],1,npix,outfile);

    (void) fclose(outfile);
    return;
}
```

APPENDIX B: HOUGH TRANSFORM SOURCE CODE

HOUGH.FOR

```
program hough
byte b_image(256),r_image(256)
integer counter,u(256),max,x_int(256,256)
integer gray_level,h_theta(256),h_rho(256)
integer i_image(256,256),accum(256,256),it,ir
real x0,y0,rho(256),x(256,256),pi/3.1415926/
real theta(256),delta_theta,rho_max,rho_0,delta_rho,factor


open(unit=1,name='test.dat',status='old',access='direct',
     *recordsize=64)
open(unit=2,name='testh.dat',status='new',access='direct',
     *recordsize=64)
open(unit=3,name='testr.dat',status='new',access='direct',
     *recordsize=64)

c     ***Begin Main Program***
c     Read in test image data.
      do j=1,256
         read(1'j) b_image
         do i=1,256
            i_image(i,j)=b_image(i)
         enddo
      enddo

      do j=1,256
         do i=1,256
         if(i_image(i,j).LT.0) i_image(i,j)=i_image(i,j)+256
         enddo
      enddo

c     ***hough transform***
      call h(i_image,accum,theta,delta_rho,rho_0,x0,y0)
      max=accum(1,1)
      min=accum(1,1)
      do j=1,256
         do i=1,256
            if(accum(i,j).GT.max) max=accum(i,j)
            if(accum(i,j).LT.min) min=accum(i,j)
         enddo
      enddo
```

```
c     Normalize accumulator array for display
c     on IM-4000 Image Manager.
      factor=243.0/(max-min)
      do j=1,256
         do i=1,256
            accum(i,j)=jnint((accum(i,j)-min)*factor)
            if(accum(i,j).GT.127) then
               b_image(i)=accum(i,j)-256
            else
               b_image(i)=accum(i,j)
            endif
         enddo
         write(2'j) b_image
      enddo

c     ***reconstruct from hough transform***
      call r(accum,max,h_theta,h_rho,rho,delta_rho,
     *       rho_0,x0,y0,x_int)

      do j=1,256
         do i=1,256
        if(x_int(i,j).GT.127) x_int(i,j)=x_int(i,j)-256
        r_image(i)=x_int(i,j)
         enddo
         write(3'j) r_image
      enddo
      end
c     ***End Main Program***

c     ******************************************************
      subroutine h(i_image,accum,theta,delta_rho,rho_0,x0,y0)
      dimension i_image(256,256)
      integer accum(256,256)
      integer gray_level
      real theta(256)
      real x0,y0
      real pi/3.1415926/

c     Initialize the accumulator array to zero.
      do j=1,256
         do i=1,256
            accum(i,j)=0
         enddo
      enddo

c     Increment theta from 0 to pi radians.
      delta_theta=pi/float(256)
      do i=1,256
         theta(i)=float(i-1)*delta_theta
      enddo
```

```
c     Fix the center of image as the origin.
      rho_max=sqrt(float(256*256)+float(256*256))
      rho_0=rho_max/2.0
      delta_rho=rho_max/float(256)
      x0=float(256/2)
      y0=float(256/2)

      do iy=1,256
         y=iy
         do ix=1,256
            gray_level=i_image(ix,iy)
            if(gray_level.LE.0) go to 5
               x=ix
               do it=1,256
c                 Hough Transformation Equation.
              rho=(x-x0)*cos(theta(it))+(y0-y)*sin(theta(it))
                  r=(rho+rho_0)/delta_rho
                  ir=jnint(r)
                  accum(it,ir)=accum(it,ir)+1
               enddo
5              continue
         enddo
      enddo
      return
      end

c     ***************************************************
      subroutine r(accum,max,h_theta,h_rho,rho,delta_rho,
     *             rho_0,x0,y0,x_int)
      integer accum(256,256),max,h_theta(256),h_rho(256)
      integer x_int(256,256),u(256),counter
      real rho(256),delta_rho,rho_0,x0,y0
      real x(256,256),pi/3.1415926/

      counter=0
      do j=1,256
         do i=1,256
            if(accum(i,j).GE.max) then
               counter=counter+1
c              Determine the theta and rho locations
c              in the accumulator array with the
c              highest count value.
               h_theta(counter)=i
               h_rho(counter)=j
            endif
         enddo
      enddo
      do j=1,counter
         rho(j)=h_rho(j)*delta_rho-rho_0
      enddo

      do j=1,counter
         do i=1,256
c           Reconstruction Equation.
            x(i,j)=x0+
     *        (rho(j)-(y0-i)*sin((h_theta(j)-1)*pi/256))/
     *         cos((h_theta(j)-1)*pi/256)
            x_int(i,j)=jnint(x(i,j))
         enddo
      enddo

      do j=1,256
         do l=1,counter
            u(l)=x_int(j,l)
         enddo

         do m=1,counter
            do k=1,256
               if(k.EQ.u(m)) x_int(k,j)=243
            enddo
         enddo
      enddo

      do j=1,256
         do i=1,256
            if(x_int(i,j).NE.243) x_int(i,j)=0
         enddo
      enddo
      return
      end
```

APPENDIX C. HEURISTIC SEARCH SOURCE CODE

HS.FOR

```
      program hs2
      byte    b_image(256,256)
      integer sample_image(256,256),clust_image(256,256),
     *        counter/0/,s_peak_row(256),s_peak_col(256),
     *        c_peak_row(256),c_peak_col(256),m,n,
     *        new_mark_row,new_mark_col,min_m,min_n,
     *        minimum_c_n,Nc,row_dist,col_dist
      real*4  d(256,256),total_cost(256),present_cost,
     *        min,factor,half_dist,d_clust(256),
     *        d_clust_total,Dc(256),minimum,row_dist_sq,
     *        col_dist_sq

      open(unit=1,name='testper5.dat',status='old',
     *     access='direct',recordsize=64)

c     ***Begin Main Program***
c     Read in test image.
      do i=1,256
         read(1'i)(b_image(i,j),j=1,256)
      enddo

c     Convert input 256x256 byte image into
c     a 256x256 integer image.
      call convert(b_image,sample_image,clust_image)

c     Locate peaks in input image.
      do i=1,256
         do j=1,256
            if(sample_image(i,j).NE.0)then
               counter=counter+1
               s_peak_row(counter)=i
               c_peak_row(counter)=i
               s_peak_col(counter)=j
               c_peak_col(counter)=j
            endif
         enddo
      enddo

c     Calculate distances between peaks.
      call dist(counter,c_peak_row,c_peak_col,d)
```

```
c     Start Main Program Loop.
      do i=1,counter
         j=(counter+1)-i
            call centroid(i,j,d,counter,min,s_peak_row,
     *                  s_peak_col,c_peak_row,c_peak_col,Dc)
            call cost(j,Dc,present_cost)
            total_cost(j)=present_cost
            print 1,j,total_cost(j)
1           format(1x,'total_cost(',i1,')=',f8.4)
      enddo

      call min_cost(total_cost,counter)
      end
c     ***End Main Program***

c     ************************************************
      subroutine convert(b_image,sample_image,clust_image)
      byte    b_image(256,256)
      integer sample_image(256,256),clust_image(256,256)
      real*4  factor

      factor=243.0/2.0
      do i=1,256
         do j=1,256
            if(b_image(i,j).LT.0)then
               sample_image(i,j)=b_image(i,j)+256
               clust_image(i,j)=sample_image(i,j)
            elseif(b_image(i,j).GE.0)then
               sample_image(i,j)=b_image(i,j)
               clust_image(i,j)=sample_image(i,j)
            endif
          sample_image(i,j)=jnint(sample_image(i,j)/factor)
          clust_image(i,j)=jnint(clust_image(i,j)/factor)
         enddo
      enddo
      return
      end

c     *********************************************
      subroutine dist(counter,c_peak_row,c_peak_col,d)
      integer counter,c_peak_row(256),c_peak_col(256)
      real*4  d(256,256)

      do m=1,counter
         do n=1,counter
          if(n.NE.m)then
      d(m,n)=sqrt(float((c_peak_row(n)-c_peak_row(m))**2)
     *          +float((c_peak_col(n)-c_peak_col(m))**2))
          endif
         enddo
      enddo
      return
      end

c     **********************************************
      subroutine centroid(i,j,d,counter,min,s_peak_row,
     *               s_peak_col,c_peak_row,c_peak_col,Dc)
      integer i,j,counter,s_peak_row(256),s_peak_col(256),
     *        c_peak_row(256),c_peak_col(256)
      real*4  d(256,256),min,Dc(256)

      if(j.EQ.counter)then
         Dc(j)=0.0
      else
         call min_dist(d,counter,min,min_m,min_n)

c        Determine midpoint between two
c        closest peaks.
         call new_marker(counter,min,min_m,min_n,
     *                   c_peak_row,c_peak_col,half_dist,
     *                   new_mark_row,new_mark_col)

         call clust_array_adj(counter,c_peak_row,
     *                        c_peak_col,new_mark_row,
     *                        new_mark_col,min_m,min_n)

         call dist_clust(j,counter,s_peak_row,s_peak_col,
     *                   c_peak_row,c_peak_col,Dc)

c        Calculate distances in adjusted image.
         call dist(counter,c_peak_row,c_peak_col,d)
      endif
      return
      end

c     **********************************************
      subroutine min_dist(d,counter,min,min_m,min_n)
      integer counter,min_m,min_n
      real*4  d(256,256),min

      do m=1,counter
         do n=1,counter
            if(d(m,n).NE.0)then
               min=d(m,n)
               goto 10
            endif
         enddo
      enddo
10    continue
```

```
      do m=1,counter
         do n=1,counter
            if(d(m,n).NE.0)then
               if(d(m,n).LE.min)then
                  min=d(m,n)
                  min_m=m
                  min_n=n
               endif
            endif
         enddo
      enddo
      return
      end

c     ***************************************************
      subroutine new_marker(counter,min,min_m,min_n,
     *                      c_peak_row,c_peak_col,half_dist,
     *                      new_mark_row,new_mark_col)
      integer counter,min_m,min_n,c_peak_row(256),
     *        c_peak_col(256),m,n,new_mark_row,new_mark_col
      real*4  min,half_dist

      m=min_m
      n=min_n
      min_m=n
      min_n=m
      half_dist=0.5*min
      if(c_peak_row(min_m).EQ.c_peak_row(min_n))then
         new_mark_row=c_peak_row(min_m)
         new_mark_col=jnint(half_dist+
     *                      float(c_peak_col(min_m)))
      elseif(c_peak_col(min_m).EQ.c_peak_col(min_n))then
         new_mark_col=c_peak_col(min_m)
         new_mark_row=jnint(half_dist+
     *                      float(c_peak_row(min_m)))
      elseif(c_peak_col(min_m).LT.c_peak_col(min_n))then
         new_mark_row=jnint(float(c_peak_row(min_m))
     *                      + float(c_peak_row(min_n)
     *                             -c_peak_row(min_m))/2.0)
         new_mark_col=jnint(half_dist+
     *                      float(c_peak_col(min_m)))
      elseif(c_peak_col(min_m).GT.c_peak_col(min_n))then
         new_mark_row=jnint(float(c_peak_row(min_m))
     *                      + float(c_peak_row(min_n)
     *                             -c_peak_row(min_m))/2.0)
         new_mark_col=jnint(float(c_peak_col(min_m))
     *                      -half_dist)
      endif
      return
      end
```

```
c     *****************************************************
      subroutine clust_array_adj(counter,c_peak_row,
     *                           c_peak_col,new_mark_row,
     *                           new_mark_col,min_m,min_n)

      integer counter,c_peak_row(256),c_peak_col(256),
     *        new_mark_row,new_mark_col,min_m,min_n,
     *        A,B,C,D,E,F

      do n=1,counter
         A=c_peak_row(n)
         B=c_peak_row(min_m)
         C=c_peak_row(min_n)
         D=c_peak_col(n)
         E=c_peak_col(min_m)
         F=c_peak_col(min_n)

if(((A.EQ.B).OR.(A.EQ.C)).AND.((D.EQ.E).OR.(D.EQ.F)))then
            c_peak_row(n)=new_mark_row
            c_peak_col(n)=new_mark_col
         endif
      enddo
      return
      end

c     ***********************************************
      subroutine dist_clust(j,counter,s_peak_row,s_peak_col,
     *                      c_peak_row,c_peak_col,Dc)

      integer j,counter,s_peak_row(256),s_peak_col(256),
     *        c_peak_row(256),c_peak_col(256),
     *        row_dist,col_dist
      real*4  d_clust(256),d_clust_total,Dc(256),
     *        row_dist_sq,col_dist_sq

      d_clust_total=0.0

      do n=1,counter
         row_dist=s_peak_row(n)-c_peak_row(n)
         col_dist=s_peak_col(n)-c_peak_col(n)
         row_dist_sq=float(row_dist*row_dist)
         col_dist_sq=float(col_dist*col_dist)
         d_clust(n)=sqrt(row_dist_sq+col_dist_sq)
         d_clust_total=d_clust(n)+d_clust_total
       enddo

      Dc(j)=d_clust_total
      return
      end
```

```
c      **********************************
      subroutine cost(j,Dc,present_cost)
      integer j,Nc
      real*4  Dc(256),present_cost,Kn/1.0/,Kd/0.1/
      Nc=j
      present_cost=Kn*float(Nc)+Kd*Dc(j)
      return
      end

c     ***************************************
      subroutine min_cost(total_cost,counter)
      integer counter,minimum_c_n
      real*4  total_cost(256),minimum
      minimum=total_cost(counter)
      do n=1,counter
         if(total_cost(n).LE.minimum)then
            minimum=total_cost(n)
            minimum_c_n=n
         endif
      enddo
      return
      end
```

APPENDIX D: MULTIPLY.FOR SOURCE CODE

MULTIPLY.FOR

```
      program multiply
c     This program takes tracks.b, the fixed length 128x512
c     byte output file from GTT, and converts it into a
c     256x256 byte array. Then the input LOFARGRAM, also a
c     256x256 byte array, is multiplied with tracks.b. The
c     resultant image is identical to tracks.b with the
c     exception that each non-zero pixel location holds the
c     value of beam00.bin at that i,j location. The output
c     file, OUTIMG.DAT, is used as input for the hough
c     transform program.

      byte b1_image(128,512),b2_image(256,256),
     *     b3_image(256,256),b4_image(256,256)


      open(unit=1,name='tracks.b',status='old',access='direct',
     *recordsize=128,maxrec=128)
      open(unit=2,name='beam00.bin',status='old',
     *access='direct',recordsize=64,maxrec=256)
      open(unit=3,file='outimg.dat',status='new',
     *access='direct',recordsize=64,maxrec=256)

      do i=1,128
        read(1'i)(b1_image(i,j),j=1,512)

         do j=1,256
            b2_image((i*2)-1,j)=b1_image(i,j)
         enddo

         do j=257,512
            b2_image(i*2,j-256)=b1_image(i,j)
         enddo
      enddo

      do i=1,256
         read(2'i)(b3_image(i,j),j=1,256)
      enddo
```

```
do i=1,256
   do j=1,256
      if(b2_image(i,j).NE.0) then
         b4_image(i,j)=b3_image(i,j)
      else
         b4_image(i,j)=0
      endif
   enddo
enddo

do i=1,256
   write(3'i)(b4_image(i,j),j=1,256)
end do

close(unit=1)
close(unit=2)
close(unit=3)

end
```

APPENDIX E: HOUGH TRANSFORM (GRAY SCALE) SOURCE CODE

HOUGH_G.FOR

```
      program hough_g

      byte    b_image(256),out_image(256,256),r_image(256)
      integer counter,u(256),max,x_int(256,256),
     *        gray_level,h_theta(256),h_rho(256),
     *        i_image(256,256),accum(256,256),
     *        accum_norm(256,256),it,ir,test_image(256,256)
      real    x0,y0,rho(256),x(256,256),pi/3.1415926/,
     *        theta(256),delta_theta,rho_max,rho_0,
     *        delta_rho,factor

      open(unit=1,name='outimg.dat',status='old',
     *     access='direct',recordsize=64)
      open(unit=2,name='gttouth16.dat',status='new',
     *     access='direct',recordsize=64)
      open(unit=3,name='gttoutr16.dat',status='new',
     *     access='direct',recordsize=64)
      open(unit=4,name='gtths16.dat',status='new',
     *     access='direct',recordsize=64)

c     ***Begin Main Program***
c     Read in test image data.
      do j=1,256
         read(1'j) b_image
         do i=1,256
            i_image(i,j)=b_image(i)
         enddo
      enddo

      do j=1,256
         do i=1,256
         if(i_image(i,j).LT.0) i_image(i,j)=i_image(i,j)+256
         enddo
      enddo
```

```
c   ***hough transform***
    call h(i_image,accum,theta,delta_rho,rho_0,x0,y0)
    max=accum(1,1)
    min=accum(1,1)
    do j=1,256
        do i=1,256
           if(accum(i,j).GT.max) max=accum(i,j)
           if(accum(i,j).LT.min) min=accum(i,j)
        enddo
    enddo

c   Normalize accumulator array for display
c   on IM-4000 Image Manager.
    factor=243.0/(max-min)
    do j=1,256
       do i=1,256
          accum_norm(i,j)=jnint((accum(i,j)-min)*factor)
          if(accum_norm(i,j).GT.127) then
             b_image(i)=accum_norm(i,j)-256
          else
             b_image(i)=accum_norm(i,j)
          endif
       enddo
       write(2'j) b_image
    enddo

c   ***reconstruct from hough transform***
    call r(accum,accum_norm,max,h_theta,h_rho,rho,
   *       delta_rho,rho_0,x0,y0,x_int)
    do j=1,256
       do i=1,256
          if(x_int(i,j).GT.127) x_int(i,j)=x_int(i,j)-256
             r_image(i)=x_int(i,j)
       enddo
       write(3'j) r_image
    enddo
    end
c   ***End Main Program***

c   *****************************************************
    subroutine h(i_image,accum,theta,delta_rho,rho_0,x0,y0)
    integer i_image(256,256),accum(256,256),gray_level
    real    theta(256),x0,y0,pi/3.1415926/

c   Increment theta from 0 to pi radians.
    delta_theta=pi/float(256)
    do i=1,256
       theta(i)=float(i-1)*delta_theta
    enddo
```

```
c   Fix the center of image as the origin.
    rho_max=sqrt(float(256*256)+float(256*256))
    rho_0=rho_max/2.0
    delta_rho=rho_max/float(256)
    x0=float(256/2)
    y0=float(256/2)
    do iy=1,256
       y=iy
       do ix=1,256
          gray_level=i_image(ix,iy)
          if(gray_level.LE.0) go to 5
             x=ix
             do it=1,256
c               Hough Transformation Equation.
     rho=(x-x0)*cos(theta(it))+(y0-y)*sin(theta(it))
                r=(rho+rho_0)/delta_rho
                ir=jnint(r)
                accum(it,ir)=accum(it,ir)+gray_level
             enddo
5            continue
       enddo
    enddo
    return
    end

c   ****************************************************
    subroutine r(accum,accum_norm,max,h_theta,h_rho,rho,
   *             delta_rho,rho_0,x0,y0,x_int)
    byte    out_image(256,256)
    integer accum(256,256),accum_norm(256,256),max,max_th,
   *        h_theta(256),h_rho(256),x_int(256,256),u(256),
   *        counter,test_image(256,256)
    real    rho(256),delta_rho,rho_0,x0,y0,x(256,256),
   *        pi/3.1415926/

    max_th=jnint(0.16*max)
    counter=0
    do j=1,256
       do i=1,256
          if(accum(i,j).GE.max_th) then
             counter=counter+1
             test_image(i,j)=accum_norm(i,j)
c            Determine the theta and rho locations
c            in the accumulator array with the
c            highest count value.
             h_theta(counter)=i
             h_rho(counter)=j
          endif
       enddo
    enddo
```

```
      do i=1,256
         do j=1,256
            if(test_image(i,j).GT.127)then
               out_image(i,j)=test_image(i,j)-256
            else
               out_image(i,j)=test_image(i,j)
            end if
         enddo
      enddo

      do i=1,256
         write(4'i)(out_image(i,j),j=1,256)
      enddo

      do j=1,counter
         rho(j)=h_rho(j)*delta_rho-rho_0
      enddo

      do j=1,counter
         do i=1,256
c           Reconstruction Equation.
            x(i,j)=x0+
     *        (rho(j)-(y0-i)*sin((h_theta(j)-1)*pi/256))/
     *         cos((h_theta(j)-1)*pi/256)
            x_int(i,j)=jnint(x(i,j))
         enddo
      enddo

      do j=1,256
         do l=1,counter
            u(l)=x_int(j,l)
         enddo

         do m=1,counter
            do k=1,256
               if(k.EQ.u(m)) x_int(k,j)=243
            enddo
         enddo
      enddo

      do j=1,256
         do i=1,256
            if(x_int(i,j).NE.243) x_int(i,j)=0
         enddo
      enddo
      return
      end
```

LIST OF REFERENCES

1. Ross, Alexander J., Automated LOFAR Tracking an Analysis of Two Algorithms, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1990.

2. Wang, Chen-Shan, Moving Object Detection by Track Analysis, Master's Thesis, Naval Postgraduate School, Monterey, California, September, 1990.

3. Wu, L.J. and Curtis, T.E., Practical Graph Partitioning Algorithms for SONAR, Proceedings of the 1987 NSUA Conference.

4. Jensen, P.A., Optimum Network Partitioning, Operation Research, Vol. 19, pp. 916-932.

5. McConnell, R.A., Draft Notes on "Graph Theoretic Tracker", Naval Research Laboratory, Washington, D.C., 1989.

6. Duda, R.O., and Hart, P.E., Use of Hough Transformation to Detect Lines and Curves in Pictures, Communication of the ACM, Vol. 15, No. 1, pp. 11-15, January, 1972.

7. Davis, L.S., Hierarchical Generalized Hough Transforms and Line-Segment Based Generalized Hough Transforms, Pattern Recognition, Vol. 15, No. 4, pp. 277-285, 1982.

8. Cowart, A.E., Snyder, W.E. and Ruedger, W.H., The Detection of Unresolved Targets using the Hough Transform, Computer Vision, Graphics, and Image Processing, Vol. 21, pp. 222-238, July, 1982.

9. IM-4000 Image Manager Manual, METSAT, Inc., 515 South Howes, Fort Collins, CO 80521, 1988.

10. LAS Users's Manual Version 4.0, Goddard Space Flight Center, Greenbelt, Maryland, September, 1987.

11. Chen,Tung-Sheng, Simulated Annealing in Sonar Track Detection, Master's Thesis, Naval Postgraduate School, Monterey, California, December, 1990.

INITIAL DISTRIBUTION LIST

		No. copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Chairman, Code EC Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5100	1
4.	Professor Chin-Lee, Code EC/Le Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5100	2
5.	Professor Murali Tummala, Code EC/Tu Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5100	1
6.	LT Vance A. Brahosky, USN PSC 473 Box 8-13 FPO AP 96349-1400	4
7.	LT A.J. Ross, USN VT 86 NAS Pensacola Pensacola, FL 32508	1
8.	Paul A. Jensen The University of Texas at Austin College of Engineering Department of Mechanical Engineering Austin, TX 78712-1063	1

Thesis
B798225 Brahosky
c.1 A combinatorial
approach to automated
LOFARGRAM analysis.